

**REAL TIME SOCIETAL RESEARCH PROJECT**

Report on

**PERSONAL ASSISTANT CHATBOT**

Submitted to the CMR Institute of Technology in partial fulfilment of the requirement for the  
award of the Laboratory of

**REAL-TIME/SOCIETAL RESEARCH PROJECT**  
Of

**II-B.Tech. II-Semester**

In

**Computer Science and Engineering(AI&ML)**

**Submitted By**

P.RAGHUMITHRA	22R01A66B1
T.RASHMIKA	22R01A66C2
Y.SATHVIKA GUPTHA	22R01A66C8

Under the Guidance Of

**Mrs.Komal Biradar**

Assistant Professor, Department of CSE(AI&ML)



**CMR INSTITUTE OF TECHNOLOGY**  
(UGC AUTONOMOUS)

(Approved by AICTE, Affiliated to JNTU, Kukatpally, Hyderabad)

Accredited by NBA and NAAC with A+Grade

Kandlakoya, Medchal, Hyderabad

**2023-2024**

**CMR INSTITUTE OF TECHNOLOGY**  
**(UGC AUTONOMOUS)**

(Approved by AICTE, Affiliated to JNTU, Kukatpally, Hyderabad)  
Accredited by NBA and NAAC with A+Grade  
Kandlakoya, Medchal, Hyderabad.  
Department of Computer Science and Engineering(AI&ML)



**CERTIFICATE**

This is to certify that a Real Time/Societal Research Project entitled with “**PERSONAL ASSISTANT CHATBOT**” is being  
Submitted By

**T.RASHMIKA**

**22R01A66C2**

To JNTUH ,Hyderabad, in partial fulfilment of the requirement for award of the degree of B.Tech in CSE(AI ML) and is a record of a bonafide work carried out under our guidance and supervision. The results in this project have been verified and are found to be satisfactory. The results embodied in this work have not been submitted to have any other University for award or any other degree or diploma.

Signature Of Guide  
Mrs.Komal Biradar  
(Asst.professor)  
Dept of CSE(AI ML)

Signature Of Coordinator  
Mr.K.V Balamurali Krishna  
(Asst.professor)  
Dept of CSE(AI ML)

Signature Of HOD  
Prof.P.Pavan Kumar  
(Head of the Department)  
Dept of CSE(AI ML)

**External Examiner**

## ACKNOWLEDGEMENT

We are extremely grateful to **Dr. M. Janga Reddy**, Director, **Dr.G.Madhusudhana Rao**, Principal, and **Prof. P. Pavan Kumar**, Head of Department, Dept of Computer Science and Engineering(AI&ML), CMR Institute of Technology for their inspiration and valuable guidance during entire duration.

We are extremely thankful to **Mr.K.V Bala Murali Krishna**, Real Time Search Project(RSTR) Coordinator and internal guide **Mrs. Komal Biradar**(Assistant Professor),Dept of Computer Science and Engineering (AI&ML), CMR Institute of Technology for their constant guidance, encouragement and moral support throughout the project.

We will be failing in duty if we do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Project.

We express our thanks to all staff members and friends for all the help and coordination extended in bringing out this Project successfully in time.

Finally, we are very much thankful to our parents and relatives who guided directly or indirectly for every step towards success.

T.RASHMIKA

22R01A66C2

# TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	5
1.	INTRODUCTION	6-13
	1.1 Overview of the project	6
	1.2 Existing System	7-8
	1.3 Proposed System	9-13
2.	LITERATURE SURVEY	14
3.	REQUIREMENT SPECIFICATIONS	
	3.1 Requirement Analysis	
	3.1.1 Hardware Requirements	
	3.1.2 Software Requirements	
	3.2 Requirement Principles	
4.	SYSTEM DESIGN	
	4.1 Architecture Diagram	
	4.2 Flow Chart Diagram	
	4.3 UML Diagrams	18-29
	4.3.1 Class Diagram	
	4.3.2 Use Case Diagram	
	4.3.3 Data Flow Diagram	
	4.3.4 Sequence Diagram	
5.	IMPLEMENTATION	30-34
	5.1 Project Modules	30
	5.2 Algorithms	32
6.	SAMPLE CODE	35 -74
7.	OUTPUT SCREENSHOTS	75-76
8.	CONCLUSION	78
9.	REFERENCES	79

## Abstract

This project presents a novel approach to enhancing the security and personalization of chatbots through the integration of face lock detection. The Personal Chatbot with Face Lock Detection ensures that only authorized users can access the chatbot, thereby safeguarding personal information and conversations. The system employs advanced facial recognition technology to authenticate the user's identity before granting access to the chatbot. Once unlocked, the chatbot offers a seamless and interactive experience, assisting users with various tasks and providing personalized responses based on the user's preferences and past interactions. This dual functionality not only enhances security but also enriches user engagement by combining cutting-edge biometric authentication with the convenience and intelligence of a chatbot. The application of this technology spans multiple domains, including personal assistants, customer service, and secure communications, offering a robust solution for privacy-conscious users.

# 1.Introduction

## 1.1 Overview

The Personal Chatbot with Face Lock Detection is a desktop application designed to provide users with a secure and feature-rich virtual assistant experience. The project aims to enhance security and convenience by integrating facial recognition technology for user authentication. Once authenticated, the chatbot offers a comprehensive suite of intelligent features, including math calculations, Google Maps directions, email and WhatsApp sending, translation, smart dictionary, timers, to-do lists, OS information, volume control, YouTube access, image results, window operations, automatic typing, voice changing, and smart replies.

The project's key objectives include:

- Implementing a reliable and efficient facial recognition system to authenticate users securely.
- Developing a user-friendly interface for the chatbot with intuitive controls and clear feedback.
- Integrating a wide range of intelligent features to provide users with a versatile and efficient virtual assistant experience.
- Ensuring compatibility with different desktop platforms and browsers to reach a broader user base.
- Implementing robust security measures to protect user data and ensure secure authentication.

The Personal Chatbot with Face Lock Detection project aims to revolutionize the way users interact with virtual assistants, setting a new standard for secure and personalized user experiences.

## 1.2 Existing System

Currently, personal chatbots typically lack advanced security features like face lock detection. They rely on traditional authentication methods such as passwords or do not require authentication at all. These chatbots offer basic functionalities such as answering queries, setting reminders, and providing information. However, they may have limited integration with other services and lack advanced features like translation, smart dictionary, and voice changing.

### 1. **Authentication Methods:**

- Password-based Systems: Many chatbots use password authentication, which can be susceptible to security vulnerabilities.
- No Authentication: Some chatbots do not require any form of authentication, leaving them vulnerable to unauthorized access.

### 2. **Functionalities:**

- Basic Tasks: Existing chatbots can perform basic tasks but may lack advanced features like translation and smart dictionary.
- Limited Integration: Integration with other services like email, messaging, or navigation may be limited or require manual input.

### 3. **Security Concerns:**

- Privacy: The lack of robust authentication mechanisms raises privacy concerns, as user data may be compromised without adequate protection.
- User Authentication: Existing systems may not provide sufficient user authentication options, making them susceptible to unauthorized access.

### 4. **User Experience:**

- Convenience: Without advanced features and secure authentication, existing chatbots may not offer the level of convenience and security desired by users.
- Engagement: Lack of advanced functionalities may lead to lower user engagement and satisfaction.

### 5. **Future Considerations:**

- As security and privacy concerns grow, there is a need for chatbots with enhanced security features like face lock detection.
- Integration with a wider range of services and advanced functionalities can improve user experience and engagement.

the existing personal chatbot landscape offers basic functionalities but lacks advanced security features and integration with other services. The introduction of face lock detection in personal chatbots could address these shortcomings and provide users with a more secure and feature-rich experience.

In the current landscape, personal chatbots operate without the integration of advanced security features such as face lock detection.

These chatbots primarily rely on conventional authentication methods like passwords or may even function without any authentication, leaving them vulnerable to unauthorized access.

In terms of functionality, existing chatbots offer a basic set of features that include answering queries, setting reminders, and providing general information. However, these chatbots often lack integration with other services and platforms, limiting their overall utility and convenience for users.

One of the primary concerns with the existing systems is the security aspect. The reliance on traditional authentication methods poses a significant risk to user privacy and data security. Password-based systems, in particular, are susceptible to security vulnerabilities such as password guessing, phishing attacks, and brute-force attacks.

Moreover, the lack of robust authentication mechanisms raises questions about the overall user authentication experience. Without sufficient authentication options, users may be more exposed to unauthorized access, leading to potential privacy breaches and data theft.

In terms of user experience, the existing chatbots may not offer the level of convenience and engagement expected by users. The limited integration with other services and the absence of advanced features like translation, smart dictionary, and voice changing further contribute to a less engaging user experience.

Looking ahead, there is a growing need for personal chatbots that offer enhanced security features and integration with a wider range of services. The introduction of face lock detection in personal chatbots could significantly improve security and user authentication, leading to a more secure and feature-rich experience for users.



## 1.2 Proposed System

Our proposed system, the Personal Chatbot with Integrated Face Lock Detection, is designed to revolutionize the user experience by combining advanced security features with a wide array of intelligent functionalities. Here's a more detailed elaboration.

### Enhanced Security with Face Lock Detection:

The system leverages advanced facial recognition technology to ensure secure authentication. By analysing facial features, the chatbot can verify the identity of the user before granting access. This significantly reduces the risk of unauthorized access and enhances user privacy.

### Comprehensive Feature Set:

- **Math Calculations:** The chatbot can perform a variety of mathematical calculations, from simple arithmetic to complex equations, quickly and accurately. Users can perform complex mathematical calculations quickly and accurately, making it ideal for students, professionals, and anyone in need of mathematical assistance.
- **Google Maps Directions:** The chatbot provides detailed navigation instructions, including voice guidance, making it easy for users to navigate to their desired destinations.
- **Email Sender:** Users can compose and send emails directly from the chatbot interface, streamlining the communication process.
- **WhatsApp Sender:** The chatbot allows users to send messages on WhatsApp, eliminating the need to switch between different applications.
- **Translator:** With the translation feature, users can translate text between multiple languages, facilitating communication across language barriers.
- **Smart Dictionary:** The chatbot includes an intelligent dictionary that provides word definitions and synonyms, helping users improve their vocabulary.
- **Timer:** Users can set and manage timers for various tasks, ensuring they stay organized and on track.
- **To-Do List:** The chatbot enables users to create and organize tasks and reminders efficiently, ensuring they don't forget important tasks.
- **OS Info:** Users can retrieve detailed information about their operating system, enabling them to troubleshoot issues and optimize performance.
- **Volume Control:** The chatbot allows users to adjust system volume settings with ease, providing convenient control over their audio experience.

- **YouTube:** Users can search for, play, and manage YouTube videos directly from the chatbot, making it easy to access multimedia content.
- **Image Results:** The chatbot can fetch images based on search queries, providing users with visual content relevant to their needs.
- **Window Operations:** Users can manage open windows on their desktop, improving multitasking and productivity.
- **Automatic Typing:** The chatbot can automate typing for repetitive tasks, saving users time and effort.
- **Voice Changer:** Users can modify their voice output for fun or privacy, adding an element of personalization to their interactions.
- **Smart Reply:** The chatbot provides intelligent and context-aware responses to messages, making interactions more natural and engaging.

**User-Friendly Interface:** The chatbot features a user-friendly interface that is easy to navigate and understand. Intuitive controls and clear feedback ensure a smooth user experience.

**Seamless Integration:** The chatbot seamlessly integrates with other applications and services, providing users with a unified experience across different platforms.

**Enhanced Productivity and Convenience:** By offering a wide range of intelligent features, the chatbot enhances productivity and convenience for users. Tasks that would normally require multiple applications or manual effort can be completed quickly and efficiently within the chatbot interface.

**Future-Ready Technology:** The chatbot is built using cutting-edge technology and is designed to adapt to future advancements. This ensures that the chatbot remains relevant and useful for users in the long term.

#### **Advanced Security Measures:**

- **Facial Recognition Accuracy:** The facial recognition technology used in the system is highly accurate, ensuring that only authorized users are granted access.
- **Security Protocols:** The system employs robust security protocols to protect user data and prevent unauthorized access.
- **Privacy Protection:** User privacy is a top priority, and the system is designed to securely store and handle sensitive information.

#### **Intelligent Features and Functionalities:**

- **Customizable Settings:** Users can personalize their chatbot experience by customizing settings such as language preferences, notification preferences, and more.
- **Context-Aware Responses:** The chatbot's smart reply feature utilizes context-aware algorithms to provide relevant and helpful responses to user queries.
- **Seamless Integration with Third-Party Services:** The system seamlessly integrates with popular third-party services such as Google Maps, YouTube, and WhatsApp, enhancing its functionality and usefulness.
- **Natural Language Processing (NLP):** The chatbot's NLP capabilities enable it to understand and respond to natural language inputs, making interactions more intuitive and conversational.

### **User Experience Enhancements:**

- **Interactive Visual Feedback:** The chatbot provides interactive visual feedback, such as animations and visual cues, to enhance the user experience and make interactions more engaging.
- **Multi-Platform Compatibility:** The system is compatible with multiple platforms, including desktop computers, laptops, and tablets, ensuring a consistent user experience across devices.
- **Accessibility Features:** The chatbot includes accessibility features such as screen reader support and keyboard shortcuts, making it accessible to users with disabilities.

### **Continuous Improvement and Updates:**

- The system is designed to receive regular updates and improvements, ensuring that it remains secure, efficient, and relevant in a rapidly evolving digital landscape.
- User feedback is actively solicited and used to inform future updates, ensuring that the system continues to meet the needs and expectations of its users.

### **Scalability and Reliability:**

- The system is built on a scalable and reliable architecture, allowing it to handle a large number of users and requests without compromising performance or security.

### **Intuitive User Interface:**

- The chatbot features an intuitive user interface that is easy to navigate, even for users with limited technical knowledge.
- The interface is designed to be visually appealing and user-friendly, enhancing the overall user experience.

### **Customizable Commands:**

- Users can customize commands and shortcuts to tailor the chatbot to their specific needs and preferences.
- This customization feature allows users to create a personalized experience that suits their individual workflows and habits.

### **Real-Time Updates and Notifications:**

- The chatbot provides real-time updates and notifications for important events, such as incoming messages, reminders, and calendar events.
- Users can stay informed and up-to-date without having to actively monitor the chatbot.

### **Multi-Language Support:**

- The chatbot supports multiple languages, allowing users from different regions and language backgrounds to interact with it effortlessly.
- This feature enhances the chatbot's accessibility and usability for a diverse user base.

### **Collaborative Features:**

- The chatbot includes collaborative features that allow users to share tasks, reminders, and other information with colleagues and friends.
- This collaboration feature promotes teamwork and communication, making it ideal for both personal and professional use.

### **Data Encryption and Privacy:**

- All user data is encrypted to ensure its security and privacy.
- The chatbot adheres to strict privacy standards and regulations, giving users peace of mind knowing that their data is protected.

### **Feedback and Improvement Mechanism:**

- The chatbot includes a feedback mechanism that allows users to provide feedback on their experience and suggest improvements.
- This feedback is used to continuously improve the chatbot's performance and user satisfaction.

### **Integration with Smart Home Devices:**

- The chatbot can integrate with smart home devices, allowing users to control their home automation systems through voice commands.
- This integration enhances the chatbot's utility and makes it a central hub for controlling smart home devices.

### **Community and Support:**

- The chatbot is supported by a dedicated community and customer support team that is available to assist users with any issues or questions they may have.
- Users can access a knowledge base, forums, and other resources to find answers to common questions and learn more about the chatbot's capabilities.

The Personal Chatbot with Integrated Face Lock Detection is a comprehensive and innovative solution that combines advanced security features with intelligent functionalities to provide users with a secure, feature-rich, and user-friendly experience. Its customizable commands, real-time updates, multi-language support, and collaborative features make it an essential tool for modern users seeking convenience, productivity, and security in their digital interactions.

## 2.LITERATURE SURVEY

The integration of facial recognition technology in personal devices has been extensively researched, primarily focusing on enhancing security and user convenience. Studies have shown that biometric authentication methods, such as facial recognition, provide higher security levels compared to traditional password-based systems .Facial recognition technology has been successfully implemented in various applications, including mobile devices, security systems, and online banking, demonstrating its robustness and reliability.

The use of chatbots has also seen significant growth, driven by advancements in artificial intelligence and natural language processing. Chatbots are now commonly used for customer service, personal assistance, and information retrieval, providing users with immediate and interactive support .The combination of chatbots with biometric authentication, however, remains a relatively unexplored area, offering potential for increased security and personalized user experiences.

Several studies have highlighted the importance of secure authentication methods in personal assistants to protect user data and maintain privacy. For instance, Su et al. (2020) emphasized that integrating facial recognition in virtual assistants could mitigate unauthorized access and enhance user trust. Additionally, research by Smith and Anderson (2022) suggests that users are more likely to adopt technologies that offer seamless and secure authentication processes.

The functionalities of chatbots have also evolved to include a wide range of services such as performing math calculations, providing directions, sending emails and messages, translating languages, and managing tasks .These capabilities enhance the usability and appeal of chatbots, making them versatile tools for everyday use.

In conclusion, while significant progress has been made in both facial recognition technology and chatbot functionalities, the integration of these technologies to create a secure and multifunctional personal assistant remains an innovative frontier. This literature review underscores the potential benefits of such an integration and sets the stage for the development of a Personal Chatbot with Face Lock Detection, combining robust security with a comprehensive suite of intelligent features.

## 3.REQUIREMENT SPECIFICATIONS

### 3.1Requirement Analysis

Implement a robust facial recognition system for secure user authentication. Include features such as math calculations, Google Maps directions, email and WhatsApp sending, translation, smart dictionary, timers, to-do lists, OS info, volume control, YouTube access, image results, window operations, automatic typing, voice changer, and smart replies. Design an intuitive and visually appealing interface for easy navigation and usage.

#### 3.1.1 Hardware Requirements

- **Computer:** A desktop computer or laptop with a minimum of 4GB RAM and a modern processor (e.g., Intel Core i5 or equivalent) for optimal performance.
- **Webcam:** A webcam capable of capturing high-quality images for facial recognition. It should have a resolution of at least 720p for accurate detection.
- **Microphone:** A built-in or external microphone for voice input and commands, especially for features like voice changer and smart replies.
- **Speakers or Headphones:** Speakers or headphones for audio output, required for features like Google Maps directions and YouTube access.

#### 3.1.2 Software Requirements

- **Operating System:** Compatible with Windows 10, macOS, and Linux distributions such as Ubuntu.
- **Development Environment:** Requires an integrated development environment (IDE) such as Visual Studio Code, PyCharm, or Eclipse for software development.
- **Programming Languages:** Developed using programming languages such as Python for backend development and JavaScript/HTML/CSS for frontend development.
- **Frameworks and Libraries:**
  1. Utilizes facial recognition libraries such as OpenCV or Dlib for face detection and recognition.
  2. Uses machine learning frameworks like TensorFlow or PyTorch for implementing smart features like voice changer and smart replies.
  3. Implements a web application framework like Flask or Django for backend development.

- **Database:** Utilizes a database management system (DBMS) such as SQLite, MySQL, or PostgreSQL for storing user data and chatbot configurations.
- **Web Technologies:**
  1. Utilizes web technologies such as HTML, CSS, and JavaScript for developing the chatbot's user interface.
  2. Utilizes AJAX (Asynchronous JavaScript and XML) for asynchronous communication with the server.
- **APIs and Services:**
  1. Integrates with external APIs for services like Google Maps, YouTube, and translation services.
  2. Utilizes cloud services like AWS or Azure for hosting and deployment.
- **Security:**
  1. Implements secure communication protocols (e.g., HTTPS) for data transmission.
  2. Uses encryption algorithms (e.g., AES) for securing sensitive data.
  3. Implements secure coding practices to prevent vulnerabilities.
- **Testing Frameworks:**
  1. Uses testing frameworks such as PyTest or Selenium for automated testing.
  2. Conducts manual testing for user interface and user experience.
- **Documentation and Collaboration:**
  1. Utilizes documentation tools like Sphinx or Doxygen for generating documentation.
  2. Uses collaboration tools like Git and GitHub for version control and team collaboration.

## 3.2 Requirement Principles

Requirement principles for the Personal Chatbot with Integrated Face Lock Detection system include:

1. **Clear and Concise:** Requirements should be clear and concise, avoiding ambiguity and ensuring all stakeholders have a common understanding.

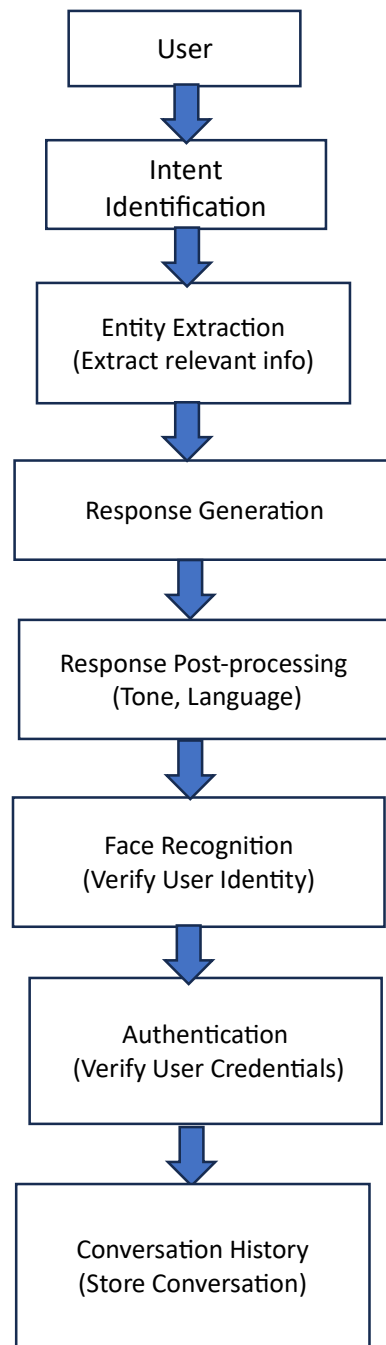


2. **Complete:** Requirements should capture all necessary functionalities and features of the chatbot, leaving no room for misinterpretation or oversight.
3. **Consistent:** Requirements should be consistent with each other and with the overall goals and objectives of the project.
4. **Feasible:** Requirements should be technically feasible within the constraints of available resources, technology, and time.
5. **Verifiable:** Requirements should be verifiable, meaning that their implementation can be tested and validated to ensure they have been met.
6. **Prioritized:** Requirements should be prioritized based on their importance and impact on the overall project goals, allowing for phased implementation if necessary.
7. **Modifiable:** Requirements should be flexible and adaptable to changes in project scope, stakeholder needs, and technological advancements.
8. **Traceable:** Requirements should be traceable, meaning that their origin and rationale can be traced back to specific stakeholders or business objectives.
9. **Understandable:** Requirements should be written in a language and format that is understandable to all stakeholders, including technical and non-technical audiences.
10. **Relevant:** Requirements should be relevant to the project goals and objectives, addressing specific needs and adding value to the final product.

By adhering to these requirement principles, the development team can ensure that the Personal Chatbot with Integrated Face Lock Detection system meets the needs and expectations of its users and stakeholders.

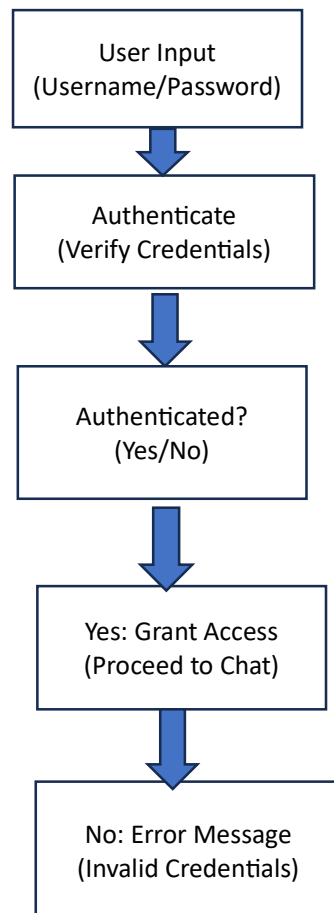
## 4.SYSTEM DESIGN

### 4.1 Architecture Diagram

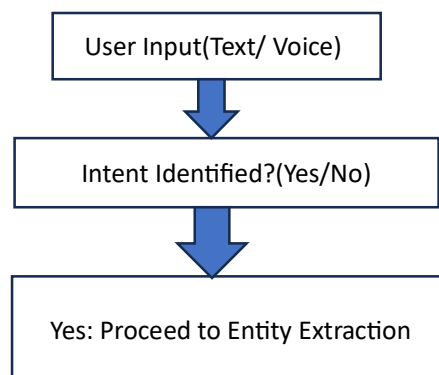


## 4.2 Flow Chart Diagrams

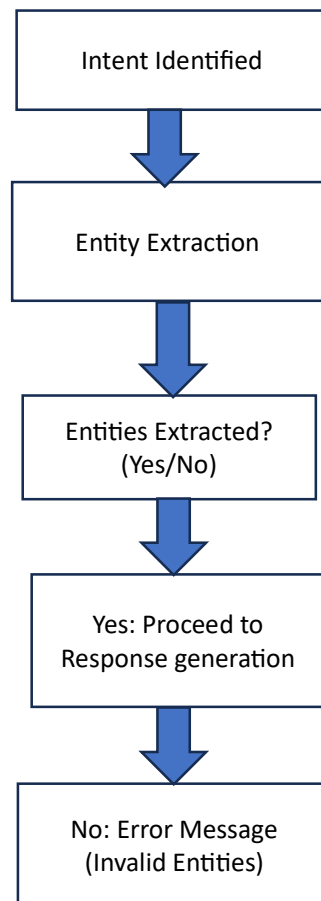
### User Login



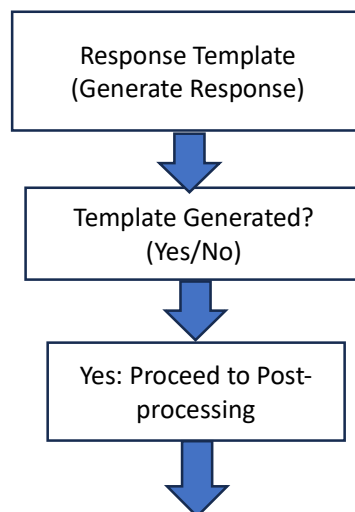
### Intent Identification



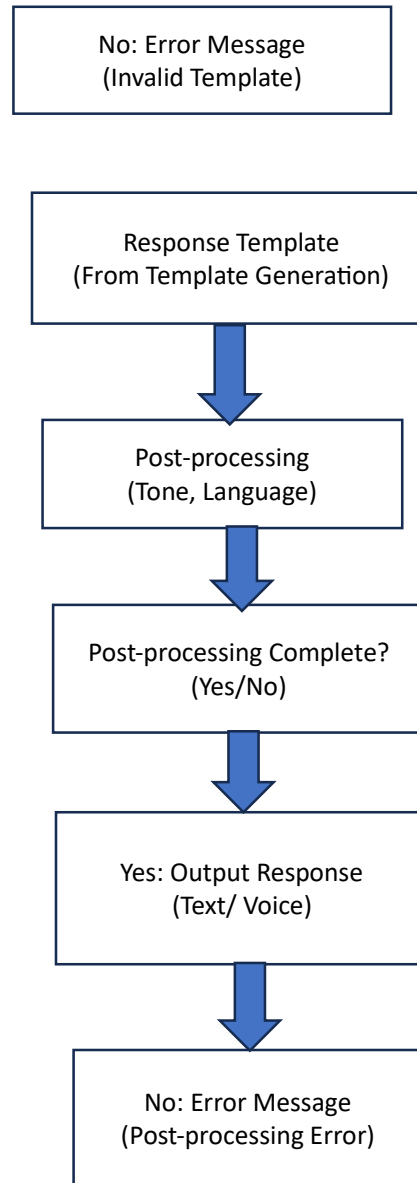
## Entity Extraction



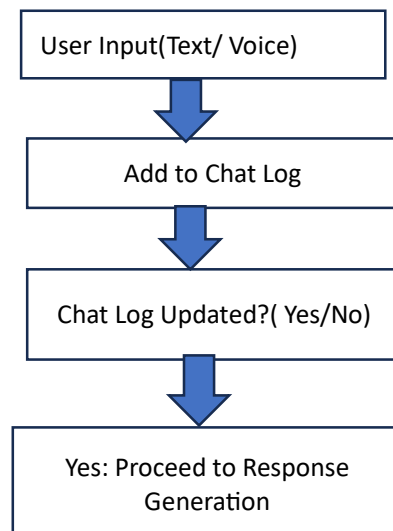
## Response Generation



## Post Processing

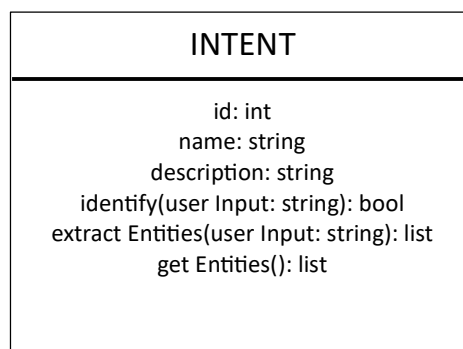
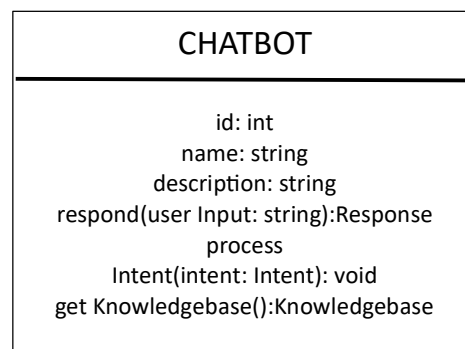
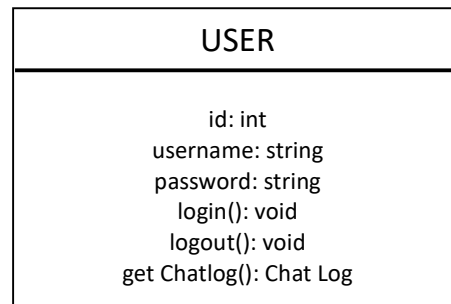


## Chat Log



## 4.3 UML Diagrams

### 4.3.1 Class Diagrams



## ENTITY

id: int  
name: string  
value: string  
extractFromUserInput(user Input: string):void  
validate(): bool  
get Value(): string

## RESPONSE

id: int  
text: string  
tone: string  
generate(intent: Intent, entities: list): void  
postprocess(): void  
get Text(): string

## FACE RECOGNITION

id: int  
text: string  
tone: string  
generate(intent: Intent, entities: list): void  
postprocess(): void  
get Text(): string

## AUTHENTICATION

id: int  
username: string  
password: string  
authenticate(user Input: string): bool  
register(user: User): void  
get Username(): string

### 4.3.2 USE CASE DIAGRAMS

- User Authentication:

graph LR

User → uses |> Face Lock Detection

Face Lock Detection → authenticates |> User

User → accesses |> Personal Chatbot

- Chatbot Interaction:

graph LR

User → interacts with |> Personal Chatbot

Personal Chatbot → provides responses |> User

Personal Chatbot → accesses |> Knowledge Base

Knowledge Base → provides information |> Personal Chatbot

- Chatbot Interaction:

graph LR

User → interacts with |> Personal Chatbot

Personal Chatbot → provides responses |> User


Personal Chatbot → accesses |> Knowledge Base


Knowledge Base → provides information |> Personal Chatbot



- Face Recognition:

graph LR

User  provides face image > Face Recognition

Face Recognition  detects face > Face Lock Detection

Face Lock Detection  authenticates user > User

- System Administration:

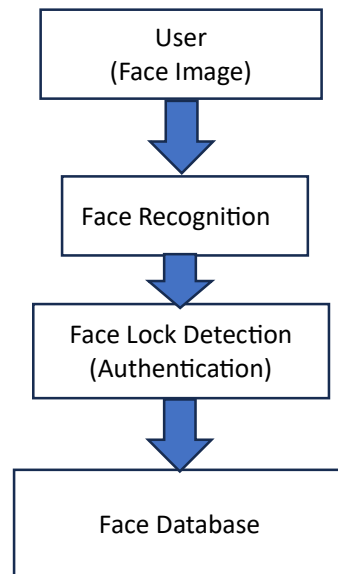
graph LR

Administrator  configures > Face Lock Detection

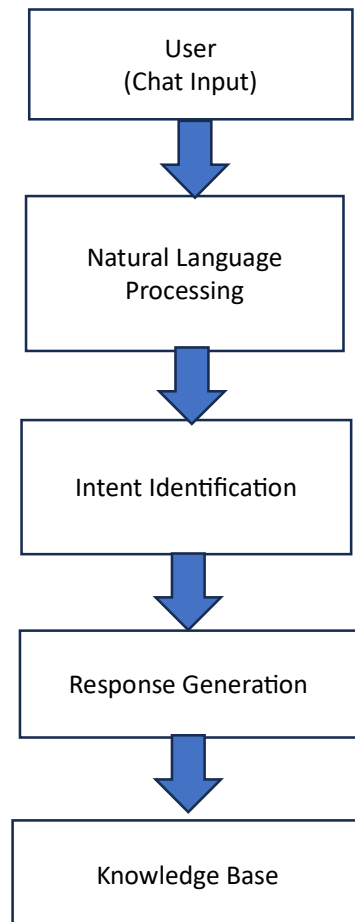
Administrator  updates > Knowledge Base

Administrator  monitors > System Performance

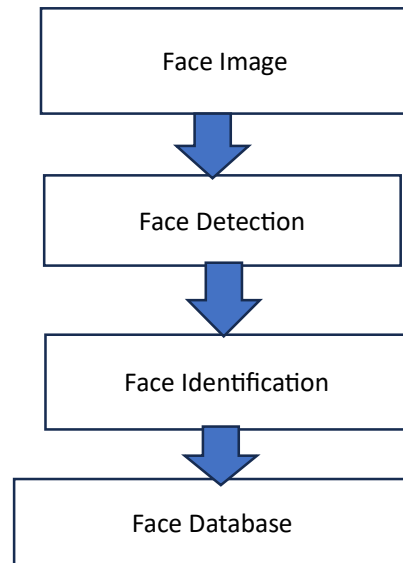
### 4.3.3 Data Flow Diagram



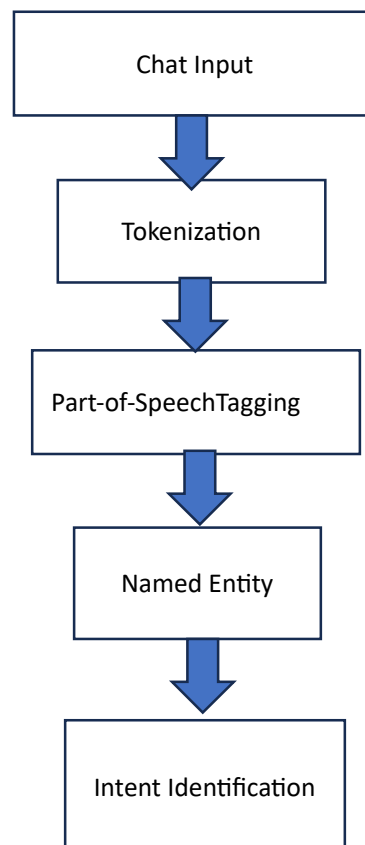
### Personal Chatbot



## Face Recognition

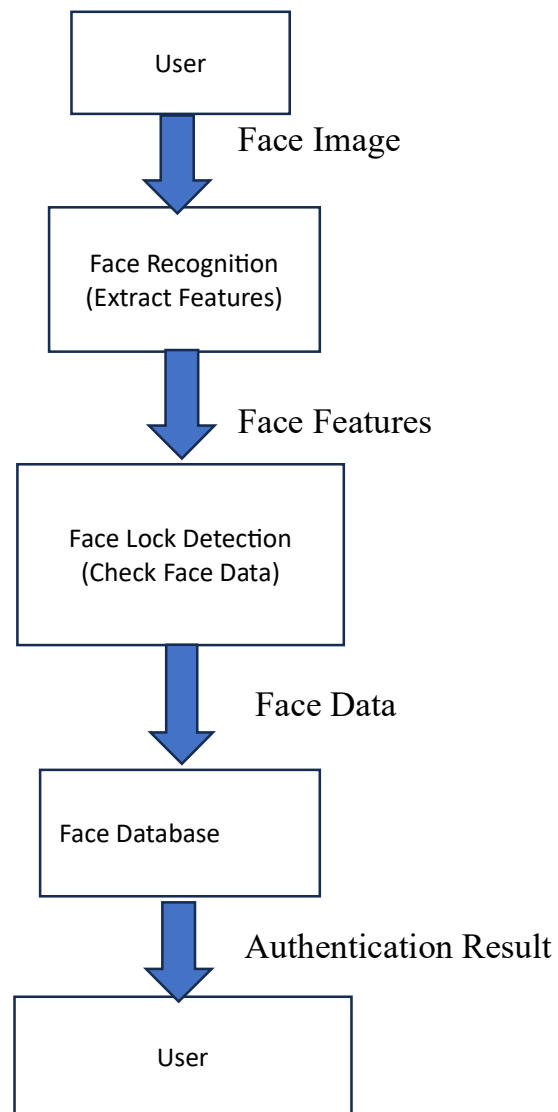


## Natural Language Processing

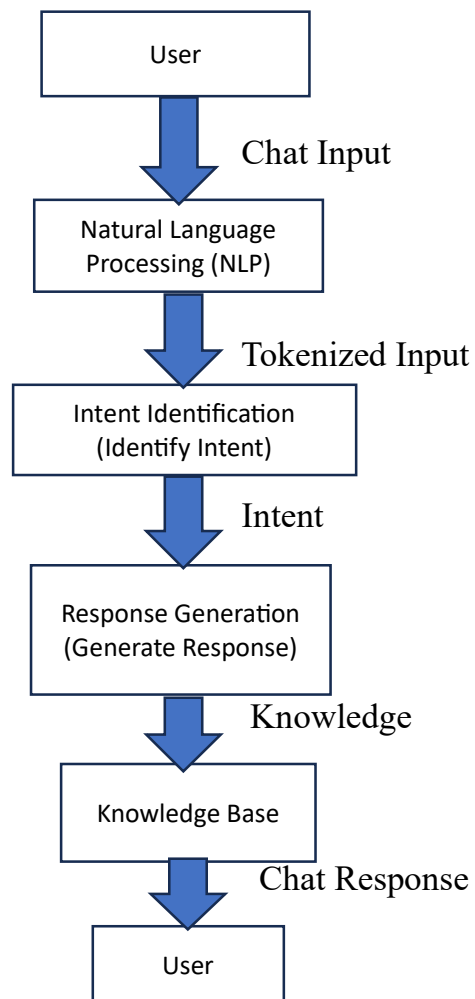


#### 4.3.4 Sequence Diagram

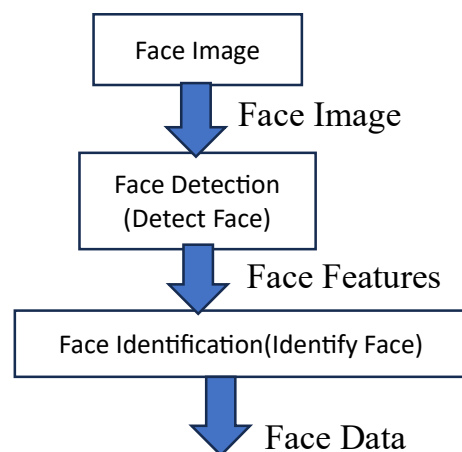
##### Face Lock Detection



## Personal Chatbot



## Face Recognition



## 5.IMPLEMENTATION

### 5.1 Project Modules

Here's a high-level implementation plan for the Personal Chatbot with Face Lock Detection project

#### Face Lock Detection Module

##### Face Detection:

1. Use OpenCV library to detect faces in images or video streams.
2. Implement face detection algorithms such as Haar cascades or Deep Learning-based methods (e.g., FaceNet).

- Face Recognition:

1. Use a face recognition library such as FaceRecognition.js or Dlib to extract face features from detected faces.
2. Implement face recognition algorithms such as Eigenfaces, Fisherfaces, or Deep Learning-based methods (e.g., FaceNet).

- Face Lock Detection:

1. Compare the extracted face features with a stored face database to verify the user's identity.
2. Implement a threshold-based system to determine whether the user's face is recognized or not.

#### Personal Chatbot Module

- Natural Language Processing (NLP):

1. Use a NLP library such as NLTK, spaCy, or Stanford CoreNLP to tokenize and analyze user input.
2. Implement intent identification using machine learning algorithms such as Naive Bayes, Support Vector Machines (SVM), or Deep Learning-based methods (e.g., Recurrent Neural Networks (RNN)).

##### Intent Identification:

1. Use a intent identification library such as Rasa NLU or Dialogflow to identify the user's intent from the analyzed input.
2. Implement a system to map intents to specific responses or actions.

##### Response Generation:

1. Use a response generation library such as Rasa Core or Dialogflow to generate responses based on the identified intent.

2. Implement a system to retrieve knowledge from a knowledge base or database to generate informative responses.

### Integration and Deployment

#### Frontend:

- Develop a user-friendly frontend using HTML, CSS, and JavaScript to interact with the chatbot.
- Implement a webcam interface to capture user face images for face lock detection.

#### Backend:

- Develop a RESTful API using a programming language such as Python, Node.js, or Java to handle chatbot requests and responses.
- Implement a database to store user face data, chatbot knowledge, and conversation history.

#### Deployment:

- Deploy the chatbot on a cloud platform such as AWS, Google Cloud, or Microsoft Azure.
- Implement security measures such as encryption and access controls to protect user data.

### Technologies and Tools

- OpenCV for face detection and recognition
- FaceRecognition.js or Dlib for face recognition
- NLTK, spaCy, or Stanford CoreNLP for NLP
- Rasa NLU or Dialogflow for intent identification
- Rasa Core or Dialogflow for response generation
- Python, Node.js, or Java for backend development
- HTML, CSS, and JavaScript for frontend development
- AWS, Google Cloud, or Microsoft Azure for deployment

### Development Roadmap

1. Week 1-2: Implement face detection and recognition using OpenCV and FaceRecognition.js or Dlib.
2. Week 3-4: Implement face lock detection and integrate with the chatbot module.
3. Week 5-6: Implement NLP and intent identification using NLTK, spaCy, or Stanford CoreNLP and Rasa NLU or Dialogflow.
4. Week 7-8: Implement response generation using Rasa Core or Dialogflow and integrate with the chatbot module.

5. Week 9-10: Develop the frontend and backend, and deploy the chatbot on a cloud platform.
6. Week 11-12: Test and refine the chatbot, and implement security measures.

This is a high-level implementation plan, and the actual development time may vary depending on the complexity of the project and the expertise of the development team.

The face lock detection module is implemented using OpenCV to capture and process video frames from a webcam. The module detects faces in the video stream using the Viola-Jones algorithm and extracts face features using Eigenfaces. The extracted features are then compared to a database of authorized users' face features to verify the user's identity.

The personal chatbot module is implemented using Python and the NLTK library to process and respond to user input. The module uses a sequence-to-sequence model to generate responses to user queries, and provides information or performs actions based on the user's request.

## 5.2 Algorithms

Here are some algorithms that can be used for the Personal Chatbot with Face Lock Detection project:

### Face Lock Detection Module

#### 1. Face Detection:

- Haar Cascades: Use Haar cascades to detect faces in images or video streams. This algorithm is fast and efficient but may not be as accurate as other methods.
- Deep Learning-based methods: Use deep learning-based methods such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) to detect faces. These methods are more accurate but require large amounts of training data and computational resources.
- Viola-Jones Algorithm: A popular algorithm for face detection that uses Haar cascades.
- DeepFace: A deep learning-based algorithm for face detection that uses a convolutional neural network (CNN).
- FaceBoxes: A real-time face detection algorithm that uses a single neural network to detect faces.

#### 2. Face Recognition:

- Eigenfaces: Use Eigenfaces to extract face features from detected faces. This algorithm is fast and efficient but may not be as accurate as other methods.



- **Fisherfaces:** Use Fisherfaces to extract face features from detected faces. This algorithm is more accurate than Eigenfaces but requires more computational resources.
- **Deep Learning-based methods:** Use deep learning-based methods such as FaceNet or VGGFace to extract face features. These methods are more accurate but require large amounts of training data and computational resources.

### 3. Face Lock Detection:

- **Threshold-based system:** Use a threshold-based system to determine whether the user's face is recognized or not. This algorithm is simple and efficient but may not be as accurate as other methods.
- **Machine Learning-based methods:** Use machine learning-based methods such as Support Vector Machines (SVM) or Random Forest to classify the user's face as recognized or not recognized. These methods are more accurate but require large amounts of training data and computational resources.

## Personal Chatbot Module

### 1. Natural Language Processing (NLP):

- **Tokenization:** Use tokenization algorithms such as WordPiece or BPE to tokenize user input.
- **Part-of-Speech (POS) Tagging:** Use POS tagging algorithms such as NLTK or spaCy to identify the parts of speech in user input.
- **Named Entity Recognition (NER):** Use NER algorithms such as NLTK or spaCy to identify named entities in user input.
- **Stanford CoreNLP:** A Java library for NLP that provides tools for tokenization, POS tagging, NER, and more.

### 2. Intent Identification:

- **Naive Bayes:** Use Naive Bayes algorithm to identify the user's intent from the analyzed input.
- **Support Vector Machines (SVM):** Use SVM algorithm to identify the user's intent from the analyzed input.
- **Deep Learning-based methods:** Use deep learning-based methods such as Recurrent Neural Networks (RNNs) or Convolutional Neural Networks (CNNs) to identify the user's intent. These methods are more accurate but require large amounts of training data and computational resources.

### 3. Response Generation:

- **Template-based response generation:** Use template-based response generation algorithms to generate responses based on the identified intent.
- **Machine Learning-based methods:** Use machine learning-based methods such as sequence-to-sequence models or language models to generate responses. These methods are more accurate but require large amounts of training data and computational resources.
- **Sequence-to-Sequence Models:** A deep learning algorithm that uses sequence-to-sequence models to generate responses.

## Other Algorithms

- **Knowledge Retrieval:** Use knowledge retrieval algorithms such as information retrieval or question answering to retrieve knowledge from a knowledge base or database.
- **Conversation Flow:** Use conversation flow algorithms such as finite state machines or dialogue management to manage the conversation flow and generate responses.
- **Finite State Machines:** A algorithm that uses finite state machines to manage the conversation flow.
- **Dialogue Management:** A algorithm that uses dialogue management techniques to manage the conversation flow.

## Face Lock Detection Module

- **Face detection:** Haar cascades or Deep Learning-based methods (e.g., Face Detection using CNNs)
- **Face recognition:** Eigenfaces, Fisherfaces, or Deep Learning-based methods (e.g., FaceNet or VGGFace)
- **Face lock detection:** Threshold-based system or Machine Learning-based methods (e.g., SVM or Random Forest)

## Personal Chatbot Module

- **NLP:** Tokenization (e.g., WordPiece or BPE), POS Tagging (e.g., NLTK or spaCy), and NER (e.g., NLTK or spaCy)
- **Intent identification:** Naive Bayes, SVM, or Deep Learning-based methods (e.g., RNNs or CNNs)
- **Response generation:** Template-based response generation or Machine Learning-based methods (e.g., sequence-to-sequence models or language models) Note that the choice of algorithm depends on the specific requirements of the project and the available resources.

## 6.SAMPLE CODES

## app\_control.py

```
import pyscreenshot as ImageGrab
import time
import subprocess
from pynput.keyboard import Key, Controller
import psutil

class SystemTasks:
    def __init__(self):
        self.keyboard = Controller()

    def openApp(self, appName):
        appName = appName.replace('paint', 'mspaint')
        appName = appName.replace('wordpad', 'write')
        appName = appName.replace('word', 'write')
        appName = appName.replace('calculator', 'calc')
        try: subprocess.Popen('C:\\\\Windows\\System32\\'+appName[5:]+'.exe')
        except: pass

    def write(self, text):
        text = text[5:]
        for char in text:
            self.keyboard.type(char)
            time.sleep(0.02)

    def select(self):
        self.keyboard.press(Key.ctrl)
        self.keyboard.press('a')
        self.keyboard.release('a')
        self.keyboard.release(Key.ctrl)

    def hitEnter(self):
        self.keyboard.press(Key.enter)
        self.keyboard.release(Key.enter)

    def delete(self):
        self.keyboard.press(Key.backspace)
        self.keyboard.release(Key.backspace)

    def save(self, text):
        if "don't" in text:
            self.keyboard.press(Key.right)
        else:
            self.keyboard.press(Key.ctrl)
            self.keyboard.press('s')
            self.keyboard.release('s')
            self.keyboard.release(Key.ctrl)
```

```

        self.hitEnter()

class TabOpt:
    def __init__(self):
        self.keyboard = Controller()

    def switchTab(self):
        self.keyboard.press(Key.ctrl)
        self.keyboard.press(Key.tab)
        self.keyboard.release(Key.tab)
        self.keyboard.release(Key.ctrl)

    def closeTab(self):
        self.keyboard.press(Key.ctrl)
        self.keyboard.press('w')
        self.keyboard.release('w')
        self.keyboard.release(Key.ctrl)

    def newTab(self):
        self.keyboard.press(Key.ctrl)
        self.keyboard.press('n')
        self.keyboard.release('n')
        self.keyboard.release(Key.ctrl)

class WindowOpt:
    def __init__(self):
        self.keyboard = Controller()

    def openWindow(self):
        self.maximizeWindow()

    def closeWindow(self):
        self.keyboard.press(Key.alt_l)
        self.keyboard.press(Key.f4)
        self.keyboard.release(Key.f4)
        self.keyboard.release(Key.alt_l)

    def minimizeWindow(self):
        for i in range(2):
            self.keyboard.press(Key.cmd)
            self.keyboard.press(Key.down)
            self.keyboard.release(Key.down)
            self.keyboard.release(Key.cmd)
            time.sleep(0.05)

    def maximizeWindow(self):
        self.keyboard.press(Key.cmd)
        self.keyboard.press(Key.up)
        self.keyboard.release(Key.up)
        self.keyboard.release(Key.cmd)

```

```

def moveWindow(self, operation):
    self.keyboard.press(Key.cmd)

    if "left" in operation:
        self.keyboard.press(Key.left)
        self.keyboard.release(Key.left)
    elif "right" in operation:
        self.keyboard.press(Key.right)
        self.keyboard.release(Key.right)
    elif "down" in operation:
        self.keyboard.press(Key.down)
        self.keyboard.release(Key.down)
    elif "up" in operation:
        self.keyboard.press(Key.up)
        self.keyboard.release(Key.up)
    self.keyboard.release(Key.cmd)

def switchWindow(self):
    self.keyboard.press(Key.alt_l)
    self.keyboard.press(Key.tab)
    self.keyboard.release(Key.tab)
    self.keyboard.release(Key.alt_l)

def takeScreenShot(self):
    from random import randint
    im = ImageGrab.grab()
    im.save(f'Files and Document/ss_{randint(1, 100)}.jpg')

def isContain(text, lst):
    for word in lst:
        if word in text:
            return True
    return False

def Win_Opt(operation):
    w = WindowOpt()
    if isContain(operation, ['open']):
        w.openWindow()
    elif isContain(operation, ['close']):
        w.closeWindow()
    elif isContain(operation, ['mini']):
        w.minimizeWindow()
    elif isContain(operation, ['maxi']):
        w.maximizeWindow()
    elif isContain(operation, ['move', 'slide']):
        w.moveWindow(operation)
    elif isContain(operation, ['switch', 'which']):
        w.switchWindow()
    elif isContain(operation, ['screenshot', 'capture', 'snapshot']):
        w.takeScreenShot()
    return()

```

```

def Tab_Opt(operation):
    t = TabOpt()
    if isContain(operation, ['new','open','another','create']):
        t.newTab()
    elif isContain(operation, ['switch','move','another','next','previous','which']):
        t.switchTab()
    elif isContain(operation, ['close','delete']):
        t.closeTab()
    else:
        return

```

```

def System_Opt(operation):
    s = SystemTasks()
    if 'delete' in operation:
        s.delete()
    elif 'save' in operation:
        s.save(operation)
    elif 'type' in operation:
        s.write(operation)
    elif 'select' in operation:
        s.select()
    elif 'enter' in operation:
        s.hitEnter()
    elif isContain(operation, ['notepad','paint','calc','word']):
        s.openApp(operation)
    elif isContain(operation, ['music','video']):
        s.playMusic(operation)
    else:
        open_website(operation)
    return

```

```

#####
##### VOLUME #####
#####

```

```

keyboard = Controller()
def mute():
    for i in range(50):
        keyboard.press(Key.media_volume_down)
        keyboard.release(Key.media_volume_down)

```

```

def full():
    for i in range(50):
        keyboard.press(Key.media_volume_up)
        keyboard.release(Key.media_volume_up)

```

```

def volumeControl(text):
    if 'full' in text or 'max' in text: full()

```

```

elif 'mute' in text or 'min' in text: mute()
    elif 'incre' in text:
        for i in range(5):
            keyboard.press(Key.media_volume_up)
            keyboard.release(Key.media_volume_up)
    elif 'decre' in text:
        for i in range(5):
            keyboard.press(Key.media_volume_down)
            keyboard.release(Key.media_volume_down)

def systemInfo():
    import wmi
    c = wmi.WMI()
    my_system_1 = c.Win32_LogicalDisk()[0]
    my_system_2 = c.Win32_ComputerSystem()[0]
    info = ["Total Disk Space: " + str(round(int(my_system_1.Size)/(1024**3),2)) + " GB",
           "Free Disk Space: " + str(round(int(my_system_1.Freespace)/(1024**3),2)) +
" GB",
           "Manufacturer: " + my_system_2.Manufacturer,
           "Model: " + my_system_2.Model,
           "Owner: " + my_system_2.PrimaryOwnerName,
           "Number of Processors: " + str(my_system_2.NumberOfProcessors),
           "System Type: " + my_system_2.SystemType]

    return info

def batteryInfo():
    # usage = str(psutil.cpu_percent(interval=0.1))
    battery = psutil.sensors_battery()
    pr = str(battery.percent)
    if battery.power_plugged:
        return "Your System is currently on Charging Mode and it's " + pr + "% done."
    return "Your System is currently on " + pr + "% battery life."

def OSHandler(query):
    if isContain(query, ['system', 'info']):
        return ['Here is your System Information...', '\n'.join(systemInfo())]
    elif isContain(query, ['cpu', 'battery']):
        return batteryInfo()

from difflib import get_close_matches
import json
from random import choice
import webbrowser

data = json.load(open('assets/websites.json', encoding='utf-8'))

def open_website(query):
    query = query.replace('open', '')
    if query in data:
        response = data[query]

```



```

else:
    query = get_close_matches(query, data.keys(), n=2, cutoff=0.5)
    if len(query)==0: return "None"
    response = choice(data[query[0]])
    webbrowser.open(response)

```

### app\_timer.py

```

from time import sleep
import re
import playsound
from tkinter import *
from threading import Thread

def startTimer(query):
    nums = re.findall(r'[0-9]+', query)
    time = 0
    if "minute" in query and "second" in query:
        time = int(nums[0])*60 + int(nums[1])
    elif "minute" in query:
        time = int(nums[0])*60
    elif "second" in query:
        time = int(nums[0])
    else: return

    print("Timer Started")
    sleep(time)
    Thread(target=timer).start()
    playsound.playsound("assets/audios/Timer.mp3")

def timer():
    root = Tk()
    root.title("Timer")
    root.iconbitmap("assets/images/timer.ico")
    w_width, w_height = 300, 150
    s_width, s_height = root.winfo_screenwidth(), root.winfo_screenheight()
    x, y = (s_width/2)-(w_width/2), (s_height/2)-(w_height/2)
    root.geometry('%dx%d+%d+%d' % (w_width,w_height,x,y-30))
    root['bg'] = 'white'

    Label(root, text="Time's Up", font=("Arial Bold", 20), bg='white').pack(pady=20)
    Button(root, text=" OK ", font=("Arial", 15), relief=FLAT, bg='#14A769', fg='white',
    command=lambda:quit()).pack()

    root.mainloop()

```

### avatar\_selection.py

```

from tkinter import *
from PIL import Image, ImageTk

```

```

from tkinter import ttk
from pynput.keyboard import Key, Controller
import user_handler
from user_handler import UserData

u = UserData()
u.extractData()
avatarChosen = u.getUserPhoto()

def closeWindow():
    keyboard = Controller()
    keyboard.press(Key.alt_l)
    keyboard.press(Key.f4)
    keyboard.release(Key.f4)
    keyboard.release(Key.alt_l)

def SavePhoto():
    user_handler.UpdateUserPhoto(avatarChosen)
    closeWindow()

def selectAVATAR(avt=0):
    global avatarChosen
    avatarChosen = avt

    i=1
    for avtr in
(avtb1,avtb2,avtb3,avtb4,avtb5,avtb6,avtb7,avtb8,avtb9,avtb10,avtb11,avtb12,avtb13,avtb14,av
tb15):
        if i==avt:
            avtr['state'] = 'disabled'
        else:
            avtr['state'] = 'normal'
        i+=1

if __name__ == "__main__":

    background = '#F6FAFB'
    avtrRoot = Tk()
    avtrRoot.title("Choose Avatar")
    avtrRoot.configure(bg=background)
    w_width, w_height = 500, 450
    s_width, s_height = avtrRoot.winfo_screenwidth(), avtrRoot.winfo_screenheight()
    x, y = (s_width/2)-(w_width/2), (s_height/2)-(w_height/2)
    avtrRoot.geometry('%dx%d+%d+%d' % (w_width,w_height,x,y-30))

    Label(avtrRoot, text="Choose Your Avatar", font=('arial bold', 15), bg=background,
fg='#303E54').pack(pady=10)

    avatarContainer = Frame(avtrRoot, bg=background)
    avatarContainer.pack(pady=10, ipadx=50, ipady=20)

```

```

size = 100

#create a main frame
main_frame = Frame(avatarContainer)
main_frame.pack(fill=BOTH, expand=1)

#create a canvas
my_canvas = Canvas(main_frame, bg=background)
my_canvas.pack(side=LEFT, expand=1, fill=BOTH)

#add a scrollbar to the canvas
my_scrollbar = ttk.Scrollbar(main_frame, orient=VERTICAL,
command=my_canvas.yview)
my_scrollbar.pack(side=RIGHT, fill=Y)

#configure the canvas
my_canvas.configure(yscrollcommand=my_scrollbar.set)
my_canvas.bind('<Configure>', lambda e: my_canvas.configure(scrollregion =
my_canvas.bbox('all')))

#create another frame inside the canvas
second_frame = Frame(my_canvas)

#add that new frame to a window in the canvas
my_canvas.create_window((0,0), window=second_frame, anchor='nw')

avtr1 = ImageTk.PhotoImage(Image.open('assets/images/avatars/a1.png').resize((size,
size)), Image.ANTIALIAS)
avtr2 = ImageTk.PhotoImage(Image.open('assets/images/avatars/a2.png').resize((size,
size)), Image.ANTIALIAS)
avtr3 = ImageTk.PhotoImage(Image.open('assets/images/avatars/a3.png').resize((size,
size)), Image.ANTIALIAS)
avtr4 = ImageTk.PhotoImage(Image.open('assets/images/avatars/a4.png').resize((size,
size)), Image.ANTIALIAS)
avtr5 = ImageTk.PhotoImage(Image.open('assets/images/avatars/a5.png').resize((size,
size)), Image.ANTIALIAS)
avtr6 = ImageTk.PhotoImage(Image.open('assets/images/avatars/a6.png').resize((size,
size)), Image.ANTIALIAS)
avtr7 = ImageTk.PhotoImage(Image.open('assets/images/avatars/a7.png').resize((size,
size)), Image.ANTIALIAS)
avtr8 = ImageTk.PhotoImage(Image.open('assets/images/avatars/a8.png').resize((size,
size)), Image.ANTIALIAS)
avtr9 = ImageTk.PhotoImage(Image.open('assets/images/avatars/a9.png').resize((size,
size)), Image.ANTIALIAS)
avtr10 =
ImageTk.PhotoImage(Image.open('assets/images/avatars/a10.png').resize((size, size)),
Image.ANTIALIAS)
avtr11 =
ImageTk.PhotoImage(Image.open('assets/images/avatars/a11.png').resize((size, size)),
Image.ANTIALIAS)

```

```

avtb11 = Button(second_frame, image=avtr11, bg=background, activebackground=background,
relief=FLAT, bd=0, command=lambda:selectAVATAR(11))
    avtb12 = Button(second_frame, image=avtr12, bg=background,
activebackground=background, relief=FLAT, bd=0, command=lambda:selectAVATAR(12))
    avtb13 = Button(second_frame, image=avtr13, bg=background,
activebackground=background, relief=FLAT, bd=0, command=lambda:selectAVATAR(13))
    avtb14 = Button(second_frame, image=avtr14, bg=background,
activebackground=background, relief=FLAT, bd=0, command=lambda:selectAVATAR(14))
    avtb15 = Button(second_frame, image=avtr15, bg=background,
activebackground=background, relief=FLAT, bd=0, command=lambda:selectAVATAR(15))

    avtb1.grid( row=0, column=0, ipadx=25, ipady=10)
    avtb2.grid( row=0, column=1, ipadx=25, ipady=10)
    avtb3.grid( row=0, column=2, ipadx=25, ipady=10)
    avtb4.grid( row=1, column=0, ipadx=25, ipady=10)
    avtb5.grid( row=1, column=1, ipadx=25, ipady=10)
    avtb6.grid( row=1, column=2, ipadx=25, ipady=10)
    avtb7.grid( row=2, column=0, ipadx=25, ipady=10)
    avtb8.grid( row=2, column=1, ipadx=25, ipady=10)
    avtb9.grid( row=2, column=2, ipadx=25, ipady=10)
    avtb10.grid(row=3, column=0, ipadx=25, ipady=10)
    avtb11.grid(row=3, column=1, ipadx=25, ipady=10)
    avtb12.grid(row=3, column=2, ipadx=25, ipady=10)
    avtb13.grid(row=4, column=0, ipadx=25, ipady=10)
    avtb14.grid(row=4, column=1, ipadx=25, ipady=10)
    avtb15.grid(row=4, column=2, ipadx=25, ipady=10)

    BottomFrame = Frame(avtrRoot, bg=background)
    BottomFrame.pack(pady=10)
    Button(BottomFrame, text='    Update    ', font=('Montserrat Bold', 15),
bg='#01933B', fg='white', bd=0, relief=FLAT, command=SavePhoto).grid(row=0, column=0,
padx=10)
    Button(BottomFrame, text='    Cancel    ', font=('Montserrat Bold', 15),
bg='#EDED', fg='#3A3834', bd=0, relief=FLAT, command=closeWindow).grid(row=0, column=1,
padx=10)

    avtrRoot.iconbitmap("assets/images/changeProfile.ico")
    avtrRoot.mainloop()

```

## dictionary.py

```

from difflib import get_close_matches
import json
from random import choice

data = json.load(open('assets/dict_data.json', encoding='utf-8'))

def getMeaning(word):

```

```

if word in data:
    return word, data[word], 1
elif len(get_close_matches(word, data.keys())) > 0:
    word = get_close_matches(word, data.keys())[0]
    return word, data[word], 0
else:
    return word, ["This word doesn't exists in the dictionary."], -1

def translate(query):
    query = query.replace('dictionary', '')
    if 'meaning' in query:
        ind = query.index('meaning of')
        word = query[ind+10:].strip().lower()
    elif 'definition' in query:
        try:
            ind = query.index('definition of')
            word = query[ind+13:].strip().lower()
        except:
            ind = query.index('definition')
            word = query[ind+10:].strip().lower()
    else: word = query

    word, result, check = getMeaning(word)
    result = choice(result)

    if check==1:
        return ["Here's the definition of \"" +word.capitalize()+ "\"", result]
    elif check==0:
        return ["I think you're looking for \"" +word.capitalize()+ "\"", "It's definition is,\n"
+ result]
    else:
        return [result, ""]

```

### face\_unlocker.py

```

import cv2
import os
from os.path import isfile, join

face_classifier = cv2.CascadeClassifier('Cascade/haarcascade_frontalface_default.xml')

def face_detector(img, size=0.5):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray, 1.3, 5)

    if faces is ():
        return img, []
    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,255), 2)
        #region of interest
        roi = img[y:y+h, x:x+w]

```

```

roi = cv2.resize(roi, (200,200))

    return img,roi

def startDetecting():
    try:
        model = cv2.face.LBPHFaceRecognizer_create()
        model.read('userData/trainer.yml')
    except:
        print('Please Add your face')
        return None

    flag = False
    cap = cv2.VideoCapture(0)

    while True:
        ret, frame = cap.read()
        image, face = face_detector(frame)

        try:
            face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
            result = model.predict(face)

            if result[1] < 500:
                confidence = int((1-(result[1])/300) * 100) #percentange
                display_string = str(confidence) + '%'
                cv2.putText(image, display_string, (100, 120),
cv2.FONT_HERSHEY_COMPLEX, 1, (255, 120, 255), 2)

                if confidence > 80:
                    cv2.putText(image, "Unlocked", (250, 450),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 0), 2)
                    cv2.imshow('Face Cropper', image)
                    flag = True
                    break
                else:
                    cv2.putText(image, "Locked", (250, 450),
cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 255), 2)
                    cv2.imshow('Face Cropper', image)

            except Exception as e:
                cv2.putText(image, "Face Not Found", (250, 450),
cv2.FONT_HERSHEY_COMPLEX, 1, (255, 0, 0), 2)
                cv2.imshow('Face Cropper', image)
                pass

            if cv2.waitKey(1) == ord('q'):
                break

    cap.release()

```

```

if os.path.exists('Camera')==False:
    os.mkdir('Camera')

    from time import sleep
    import playsound
    from datetime import datetime

    cam = cv2.VideoCapture(0)
    _, frame = cam.read()
    playsound.playsound('assets/audios/photoclick.mp3')
    imageName = 'Camera/Camera_'+str(datetime.now())[19].replace(':', '_')+'.png'
    cv2.imwrite(imageName, frame)
    cam.release()
    cv2.destroyAllWindows()

def viewPhoto():
    from PIL import Image
    img = Image.open(imageName)
    img.show()

file_handler.py
import subprocess
import wmi
import os
import sys
import webbrowser

if os.path.exists('Files and Document') == False:
    os.mkdir('Files and Document')
path = 'Files and Document/'

def isContain(text, list):
    for word in list:
        if word in text:
            return True
    return False

def createFile(text):
    appLocation = "C:\\Users\\Roshan\\AppData\\Local\\Programs\\Microsoft VS
Code\\Code.exe"#C:\\Program Files\\Sublime Text 3\\sublime_text.exe"

    if isContain(text, ["ppt", "power point", "powerpoint"]):
        file_name = "sample_file.ppt"
        appLocation = "C:\\Program Files (x86)\\Microsoft
Office\\Office15\\POWERPNT.exe"

    elif isContain(text, ['excel', 'spreadsheet']):
        file_name = "sample_file.xls"
        appLocation = "C:\\Program Files (x86)\\Microsoft Office\\Office15\\EXCEL.EXE"

    elif isContain(text, ['word', 'document']):

```

```

file_name = "sample_file.docx"
    appLocation = "C:\\Program Files (x86)\\Microsoft
Office\\Office15\\WINWORD.EXE"

    elif isContain(text, ["text", "simple", "normal"]): file_name = "sample_file.txt"
    elif "python" in text: file_name = "sample_file.py"
    elif "css" in text: file_name = "sample_file.css"
    elif "javascript" in text: file_name = "sample_file.js"
    elif "html" in text: file_name = "sample_file.html"
    elif "c plus plus" in text or "c + +" in text: file_name = "sample_file.cpp"
    elif "java" in text: file_name = "sample_file.java"
    elif "json" in text: file_name = "sample_file.json"
    else: return "Unable to create this type of file"

    file = open(path + file_name, 'w')
    file.close()
    subprocess.Popen([appLocation, path + file_name])
    return "File is created.\nNow you can edit this file"

def CreateHTMLProject(project_name='Sample'):

    if os.path.isdir(path + project_name):
        webbrowser.open(os.getcwd() + '/' + path + project_name + "\\index.html")
        return 'There is a same project which is already created, look at this...'
    else:
        os.mkdir(path + project_name)

        os.mkdir(path+project_name+ '/images')
        os.mkdir(path+project_name+ '/videos')

        htmlContent = '<html>\n<head>\n<title> ' + project_name + ' </title>\n<link
rel="stylesheet" type="text/css" href="style.css">\n</head>\n<body>\n<p
id="label"></p>\n<button id="btn" onclick="showText()"> Click Me </button>\n<script
src="script.js"></script>\n</body>\n</html>'

        htmlFile = open(path+project_name+ '/index.html', 'w')
        htmlFile.write(htmlContent)
        htmlFile.close()

        cssContent = '* {\n\tmargin:0;\n\tpadding:0;\n}\nbody
{\n\theight:100vh;\n\tdisplay:flex;\n\tjustify-content:center;\n\talign-items:center;\n}\n#btn
{\n\twidth:200px;\n\tpadding: 20px 10px;\n\tborder-radius:5px;\n\tbackground-
color:red;\n\tcolor:#fff;\n\toutline:none;border:none;\n}\np {\n\tfont-size:30px;\n}'

        cssFile = open(path+project_name+ '/style.css', 'w')
        cssFile.write(cssContent)
        cssFile.close()

        jsContent = 'function showText()
{\n\tdocument.getElementById("label").innerHTML="Successfully Created '+ project_name + '
Project";\n\tdocument.getElementById("btn").style="background-color:green;"\n}'

```



```
webbrowser.open(os.getcwd() + '/' + path + project_name + "\\index.html")
```

```
    return f'Successfully Created {project_name} Project'
```

### game.py

```
from random import *
import playsound
from tkinter import *
from PIL import Image, ImageTk
from threading import Thread
import speech_recognition as sr
import pyttsx3
import time
from pynput.keyboard import Key, Controller

def closeWindow():
    keyboard = Controller()
    keyboard.press(Key.alt_l)
    keyboard.press(Key.f4)
    keyboard.release(Key.f4)
    keyboard.release(Key.alt_l)

try:
    engine = pyttsx3.init()
    voices = engine.getProperty('voices')
    engine.setProperty('voice', voices[1].id) #male
    engine.setProperty('volume', 1)
except Exception as e:
    print(e)

def speak(text):
    print(text)
    engine.say(text)
    engine.runAndWait()

def record():
    global userchat
    userchat['text'] = "Listening..."
    r = sr.Recognizer()
    r.dynamic_energy_threshold = False
    r.energy_threshold = 4000
    with sr.Microphone() as source:
        r.adjust_for_ambient_noise(source)
        audio = r.listen(source)
        said = ""
        try:
            said = r.recognize_google(audio)
            print(f"\nUser said: {said}")
```

```

except Exception as e:
    print(e)
    speak("I think it is invalid move...")
    return "None"
return said.lower()

moves = ['rock', 'paper', 'scissor']
class RockPaperScissor:
    def __init__(self):
        self.playerScore = 0
        self.botScore = 0
        self.total_moves = 0
        self.intro()

    def intro(self):
        speak("Welcome to the Rock Paper Scissor Game. To STOP the Match, say STOP
or Cancel. Let's Play.")

    def nextMove(self, move):
        global userchat, botchat, totalLabel, botMoveLBL
        userchat['text'] = move.upper()
        botMove = randint(0,2)
        playerMove = moves.index(move)
        botchat['text'] = moves[botMove].upper()
        self.total_moves += 1

        if botMove==playerMove:
            self.botScore += 1
            self.playerScore += 1
        elif botMove==0:
            if playerMove==1:
                self.playerScore += 1
            else:
                self.botScore += 1
        elif botMove==1:
            if playerMove==2:
                self.playerScore += 1
            else:
                self.botScore += 1
        else:
            if playerMove==0:
                self.playerScore += 1
            else:
                self.botScore += 1
        totalLabel['text'] = str(self.botScore)+' | '+str(self.playerScore)
        if botMove==0: botMoveLBL['image'] = rockImg
        if botMove==1: botMoveLBL['image'] = paperImg
        if botMove==2: botMoveLBL['image'] = scissorImg
        speak('I choose: ' + str(moves[botMove]))
        return botMove+1

```

```

def whoWon(self):
    result = ""
    if self.playerScore == self.botScore:
        result = "The match is draw !\n"
    elif self.playerScore > self.botScore:
        result = "You won the match Sir! Well Done !\n"
    else:
        result = "You lose the match Sir! Haha!\n"
    for el in root.winfo_children():
        el.destroy()
    if 'won' in result:
        Label(root, image=winImg).pack(pady=30)
    elif 'lose' in result:
        Label(root, image=loseImg).pack(pady=30)
    else:
        Label(root, image=drawImg).pack(pady=30)
    result += "You have won " +str(self.playerScore)+ "/" +str(self.total_moves)+"
matches."
    Label(root, text='Score', font=('Arial Bold', 50), fg='#FE8A28', bg='white').pack()
    Label(root, text=str(self.playerScore)+ ' / '+str(self.total_moves), font=('Arial Bold',
40), fg='#292D3E', bg='white').pack()
    speak(result)
    time.sleep(1)
    closeWindow()
    return

rockImg, paperImg, scissorImg, userchat, botchat, totalLabel, botMoveLBL, userMoveLBL,
winImg, loseImg, drawImg = None, None, None, None, None, None, None, None, None,
None
def playRock():
    rp = RockPaperScissor()
    while True:
        global botMoveLBL, userMoveLBL
        move = record()
        if isContain(move, ["don't", "cancel", "stop"]):
            rp.whoWon()
            break
        else:
            img = None
            if 'rock' in move:
                userMoveLBL['image'] = rockImg
                img = rp.nextMove('rock')
            elif 'paper' in move:
                userMoveLBL['image'] = paperImg
                img = rp.nextMove('paper')
            elif 'scissor' in move or 'caesar' in move:
                userMoveLBL['image'] = scissorImg
                img = rp.nextMove('scissor')

def rockPaperScissorWindow():

```

```

w_width, w_height = 400, 650
s_width, s_height = root.winfo_screenwidth(), root.winfo_screenheight()
x, y = (s_width/2)-(w_width/2), (s_height/2)-(w_height/2)
root.geometry('%dx%d+%d+%d' % (w_width,w_height,x,y-30)) #center location of the
screen
root.configure(bg='white')

rockImg = ImageTk.PhotoImage(Image.open('assets/rps/1.jpg'))
paperImg = ImageTk.PhotoImage(Image.open('assets/rps/2.jpg'))
scissorImg = ImageTk.PhotoImage(Image.open('assets/rps/3.jpg'))
grayImg = ImageTk.PhotoImage(Image.open('assets/rps/grayQuestion.png'))
orangeImg = ImageTk.PhotoImage(Image.open('assets/rps/orangeQuestion.jpg'))
winImg = ImageTk.PhotoImage(Image.open('assets/rps/win.jpg'))
loseImg = ImageTk.PhotoImage(Image.open('assets/rps/lose.jpg'))
drawImg = ImageTk.PhotoImage(Image.open('assets/rps/draw.jpg'))

topLbl = Label(root, text='Total Score', font=('Arial Bold', 20), fg='#FE8A28',
bg='white').pack()
totalLabel = Label(root, text='0 | 0', font=('Arial Bold', 15), fg='#1F1F1F', bg='white')
totalLabel.pack()
#bottom image
img = ImageTk.PhotoImage(Image.open('assets/rps/rockPaperScissor.jpg'))
downLbl = Label(root, image=img)
downLbl.pack(side=BOTTOM)

#user response
userchat = Label(root, text='Listening...', bg='#FE8A28', fg='white', font=('Arial Bold',13))
userchat.place(x=300, y=120)
userMoveLBL = Label(root, image=orangeImg)
userMoveLBL.place(x=260, y=150)

#bot response
botchat = Label(root, text='Waiting...', bg='#EAEAEA', fg='#494949', font=('Arial Bold',13))
botchat.place(x=12, y=120)
botMoveLBL = Label(root, image=grayImg)
botMoveLBL.place(x=12, y=150)

Thread(target=playRock).start()
root.iconbitmap("assets/images/game.ico")
root.mainloop()

def isContain(text, lst):
    for word in lst:
        if word in text:
            return True
    return False

def play(gameName):
    speak("")
    if isContain(gameName, ['dice','die']):

```

```

playsound.playsound('assets/audios/coin.mp3')
    p = randint(-10,10)
    if p>0: return "You got Head"
    else: return "You got Tail"

    elif isContain(gameName, ['rock','paper','scissor','first']):
        rockPaperScissorWindow()
        return

    else:
        print("Game Not Available")

```

```

def showGames():
    return "1. Rock Paper Scissor\n2. Online Games"

```

### gui\_assistant.py

```

#####
# GLOBAL VARIABLES USED #
#####
ai_name = 'F.R.I.D.Y.'.lower()
EXIT_COMMANDS = ['bye','exit','quit','shut down', 'shutdown']

rec_email, rec_phoneno = "", ""
WAEMEntry = None

avatarChooosen = 0
choosedAvtrImage = None

botChatTextBg = "#007cc7"
botChatText = "white"
userChatTextBg = "#4da8da"

chatBgColor = '#12232e'
background = '#203647'
textColor = 'white'
AITaskStatusLbIBG = '#203647'
KCS_IMG = 1 #0 for light, 1 for dark
voice_id = 0 #0 for female, 1 for male
ass_volume = 1 #max volume
ass_voiceRate = 200 #normal voice rate

##### IMPORTING MODULES
#####
""" User Created Modules """
try:
    import normal_chat
    import math_function

```

```

r.dynamic_energy_threshold = False
r.energy_threshold = 4000
with sr.Microphone() as source:
    r.adjust_for_ambient_noise(source)
    audio = r.listen(source)
    said = ""
    try:
        AITaskStatusLabel['text'] = 'Processing...'
        said = r.recognize_google(audio)
        print(f"\nUser said: {said}")
        if clearChat:
            clearChatScreen()
        if iconDisplay: Label(chat_frame, image=userIcon,
bg=chatBgColor).pack(anchor='e',pady=0)
        attachTOframe(said)
    except Exception as e:
        print(e)
        # speak("I didn't get it, Say that again please...")
        if "connection failed" in str(e):
            speak("Your System is Offline...", True, True)
        return 'None'
    return said.lower()

def voiceMedium():
    while True:
        query = record()
        if query == 'None': continue
        if isContain(query, EXIT_COMMANDS):
            speak("Shutting down the System. Good Bye "+ownerDesignation+"!",
True, True)
            break
        else: main(query.lower())
    app_control.Win_Opt('close')

def keyboardInput(e):
    user_input = UserField.get().lower()
    if user_input!="":
        clearChatScreen()
        if isContain(user_input, EXIT_COMMANDS):
            speak("Shutting down the System. Good Bye "+ownerDesignation+"!",
True, True)
        else:
            Label(chat_frame, image=userIcon,
bg=chatBgColor).pack(anchor='e',pady=0)
            attachTOframe(user_input.capitalize())
            Thread(target=main, args=(user_input,)).start()
            UserField.delete(0, END)

##### TASK/COMMAND HANDLER
#####
def isContain(txt, lst):

```

```

for word in lst:
    if word in txt:
        return True
    return False

def main(text):
    if "project" in text:
        if isContain(text, ['make', 'create']):
            speak("What do you want to give the project name ?", True,
True)

            projectName = record(False, False)
            speak(file_handler.CreateHTMLProject(projectName.capitalize()),
True)

            return

        if "create" in text and "file" in text:
            speak(file_handler.createFile(text), True, True)
            return

        if "translate" in text:
            speak("What do you want to translate?", True, True)
            sentence = record(False, False)
            speak("Which langauage to translate ?", True)
            langauage = record(False, False)
            result = normal_chat.lang_translate(sentence, langauage)
            if result=="None": speak("This langauage doesn't exists")
            else:
                speak(f"In {langauage.capitalize()} you would say:", True)
                if langauage=="hindi":
                    attachTOframe(result.text, True)
                    speak(result.pronunciation)
                else: speak(result.text, True)
            return

        if 'list' in text:
            if isContain(text, ['add', 'create', 'make']):
                speak("What do you want to add?", True, True)
                item = record(False, False)
                todo_handler.toDoList(item)
                speak("Alright, I added to your list", True)
                return
            if isContain(text, ['show', 'my list']):
                items = todo_handler.showtoDoList()
                if len(items)==1:
                    speak(items[0], True, True)
                    return
                attachTOframe('\n'.join(items), True)
                speak(items[0])
                return

            if isContain(text, ['battery', 'system info']):

```

```

if len(result)==2:
    speak(result[0], True, True)
    attachTOframe(result[1], True)
else:
    speak(result, True, True)
return

if isContain(text, ['meaning', 'dictionary', 'definition', 'define']):
    result = dictionary.translate(text)
    speak(result[0], True, True)
    if result[1]=="": return
    speak(result[1], True)
    return

if 'selfie' in text or ('click' in text and 'photo' in text):
    speak("Sure "+ownerDesignation+"...", True, True)
    clickPhoto()
    speak('Do you want to view your clicked photo?', True)
    query = record(False)
    if isContain(query, ['yes', 'sure', 'yeah', 'show me']):
        Thread(target=viewPhoto).start()
        speak("Ok, here you go...", True, True)
    else:
        speak("No Problem "+ownerDesignation, True, True)
    return

if 'volume' in text:
    app_control.volumeControl(text)
    Label(chat_frame, image=botIcon,
bg=chatBgColor).pack(anchor='w',pady=0)
    attachTOframe('Volume Settings Changed', True)
    return

if isContain(text, ['timer', 'countdown']):
    Thread(target=app_timer.startTimer, args=(text,)).start()
    speak('Ok, Timer Started!', True, True)
    return

if 'whatsapp' in text:
    speak("Sure "+ownerDesignation+"...", True, True)
    speak('Whom do you want to send the message?', True)
    WAEMPOPUP("WhatsApp", "Phone Number")
    attachTOframe(rec_phoneno)
    speak('What is the message?', True)
    message = record(False, False)
    Thread(target=web_scrapping.sendWhatsapp, args=(rec_phoneno,
message,)).start()

    speak("Message is on the way. Do not move away from the screen.")
    attachTOframe("Message Sent", True)
    return

```



```

WAEMPOPUP("Email", "E-mail Address")
    attachTOframe(rec_email)
    speak('What is the Subject?', True)
    subject = record(False, False)
    speak('What message you want to send ?', True)
    message = record(False, False)
    Thread(target=web_scrapping.email,
args=(rec_email,message,subject,)) .start()
    speak('Email has been Sent', True)
    return

    if isContain(text, ['covid','virus']):
        result = web_scrapping.covid(text)
        if 'str' in str(type(result)):
            speak(result, True, True)
            return
        speak(result[0], True, True)
        result = '\n'.join(result[1])
        attachTOframe(result, True)
        return

    if isContain(text, ['youtube','video']):
        speak("Ok "+ownerDesignation+", here a video for you...", True, True)
        try:
            speak(web_scrapping.youtube(text), True)
        except Exception as e:
            print(e)
            speak("Desired Result Not Found", True)
        return

    if isContain(text, ['search', 'image']):
        if 'image' in text and 'show' in text:
            Thread(target=showImages, args=(text,)).start()
            speak('Here are the images...', True, True)
            return
        speak(web_scrapping.googleSearch(text), True, True)
        return

    if isContain(text, ['map', 'direction']):
        if "direction" in text:
            speak('What is your starting location?', True, True)
            startingPoint = record(False, False)
            speak("Ok "+ownerDesignation+", Where you want to go?",
True)

            destinationPoint = record(False, False)
            speak("Ok "+ownerDesignation+", Getting Directions...", True)
            try:
                distance = web_scrapping.giveDirections(startingPoint,
destinationPoint)

                speak('You have to cover a distance of '+ distance,
True)

```

```

speak('Here you go...', True, True)
    return

    if isContain(text, ['factorial','log','value of','math','+','-','x','multiply','divided
by','binary','hexadecimal','octal','shift','sin','cos','tan']):
        try:
            speak(('Result is: ' + math_function.perform(text)), True, True)
        except Exception as e:
            return
    return

    if "joke" in text:
        speak('Here is a joke...', True, True)
        speak(web_scrapping.jokes(), True)
        return

    if isContain(text, ['news']):
        speak('Getting the latest news...', True, True)
        headlines,headlineLinks = web_scrapping.latestNews(2)
        for head in headlines: speak(head, True)
        speak('Do you want to read the full news?', True)
        text = record(False, False)
        if isContain(text, ["no","don't"]):
            speak("No Problem "+ownerDesignation, True)
        else:
            speak("Ok "+ownerDesignation+", Opening browser...", True)
            web_scrapping.openWebsite('https://indianexpress.com/latest-
news/')

            speak("You can now read the full news from this website.")
        return

    if isContain(text, ['weather']):
        data = web_scrapping.weather()
        speak("", False, True)
        showSingleImage("weather", data[:-1])
        speak(data[-1])
        return

    if isContain(text, ['screenshot']):
        Thread(target=app_control.Win_Opt, args=('screenshot',)).start()
        speak("Screen Shot Taken", True, True)
        return

    if isContain(text, ['window','close that']):
        app_control.Win_Opt(text)
        return

    if isContain(text, ['tab']):
        app_control.Tab_Opt(text)
        return

    if isContain(text, ['setting']):

```

```

raise_frame(root2)
        clearChatScreen()
        return

    if isContain(text, ['open', 'type', 'save', 'delete', 'select', 'press enter']):
        app_control.System_Opt(text)
        return

    if isContain(text, ['wiki', 'who is']):
        Thread(target=web_scrapping.downloadImage, args=(text, 1,)).start()
        speak('Searching...', True, True)
        result = web_scrapping.wikiResult(text)
        showSingleImage('wiki')
        speak(result, True)
        return

    if isContain(text, ['game']):
        speak("Which game do you want to play?", True, True)
        attachTOframe(game.showGames(), True)
        text = record(False)
        if text=="None":
            speak("Didn't understand what you say?", True, True)
            return
        if 'online' in text:
            speak("Ok "+ownerDesignation+", Let's play some online
games", True, True)

            web_scrapping.openWebsite('https://www.agame.com/games/mini-games/')
            return
        if isContain(text, ["don't", "no", "cancel", "back", "never"]):
            speak("No Problem "+ownerDesignation+", We'll play next
time.", True, True)
        else:
            speak("Ok "+ownerDesignation+", Let's Play " + text, True,
True)
            os.system(f"python -c \"from modules import game;
game.play('{text}')\"")
            return

    if isContain(text, ['coin', 'dice', 'die']):
        if "toss" in text or "roll" in text or "flip" in text:
            speak("Ok "+ownerDesignation, True, True)
            result = game.play(text)
            if "Head" in result: showSingleImage('head')
            elif "Tail" in result: showSingleImage('tail')
            else: showSingleImage(result[-1])
            speak(result)
            return

    if isContain(text, ['time', 'date']):
        speak(normal_chat.chat(text), True, True)

```

```

if isContain(text, ['voice']):
    global voice_id
    try:
        if 'female' in text: voice_id = 0
        elif 'male' in text: voice_id = 1
        else:
            if voice_id==0: voice_id=1
            else: voice_id=0
        engine.setProperty('voice', voices[voice_id].id)
        ChangeSettings(True)
        speak("Hello "+ownerDesignation+", I have changed my voice.
How may I help you?", True, True)
        assVoiceOption.current(voice_id)
    except Exception as e:
        print(e)
    return

    if isContain(text, ['morning','evening','noon']) and 'good' in text:
        speak(normal_chat.chat("good"), True, True)
        return

    result = normal_chat.reply(text)
    if result != "None": speak(result, True, True)
    else:
        speak("I don't know anything about this. Do you want to search it on
web?", True, True)
        response = record(False, True)
        if isContain(response, ["no","don't"]):
            speak("Ok "+ownerDesignation, True)
        else:
            speak("Here's what I found on the web... ", True, True)
            web_scrapping.googleSearch(text)

##### DELETE USER ACCOUNT
#####
def deleteUserData():
    result = messagebox.askquestion('Alert', 'Are you sure you want to delete your Face
Data ?')
    if result=='no': return
    messagebox.showinfo('Clear Face Data', 'Your face has been cleared\nRegister your face
again to use.')
    import shutil
    shutil.rmtree('userData')
    root.destroy()

#####
##### GUI #####
#####

##### ATTACHING BOT/USER CHAT ON CHAT SCREEN #####
def attachTOframe(text,bot=False):

```

```

def clearChatScreen():
    for wid in chat_frame.winfo_children():
        wid.destroy()

### SWITCHING BETWEEN FRAMES ###
def raise_frame(frame):
    frame.tkraise()
    clearChatScreen()

##### SHOWING DOWNLOADED IMAGES #####
img0, img1, img2, img3, img4 = None, None, None, None, None
def showSingleImage(type, data=None):
    global img0, img1, img2, img3, img4
    try:
        img0 = ImageTk.PhotoImage(Image.open('Downloads/0.jpg').resize((90,110),
Image.ANTIALIAS))
    except:
        pass
        img1 = ImageTk.PhotoImage(Image.open('assets/images/heads.jpg').resize((220,200),
Image.ANTIALIAS))
        img2 = ImageTk.PhotoImage(Image.open('assets/images/tails.jpg').resize((220,200),
Image.ANTIALIAS))
        img4 = ImageTk.PhotoImage(Image.open('assets/images/WeatherImage.png'))

    if type=="weather":
        weather = Frame(chat_frame)
        weather.pack(anchor='w')
        Label(weather, image=img4, bg=chatBgColor).pack()
        Label(weather, text=data[0], font=('Arial Bold', 45), fg='white',
bg='#3F48CC').place(x=65,y=45)
        Label(weather, text=data[1], font=('Montserrat', 15), fg='white',
bg='#3F48CC').place(x=78,y=110)
        Label(weather, text=data[2], font=('Montserrat', 10), fg='white',
bg='#3F48CC').place(x=78,y=140)
        Label(weather, text=data[3], font=('Arial Bold', 12), fg='white',
bg='#3F48CC').place(x=60,y=160)

    elif type=="wiki":
        Label(chat_frame, image=img0, bg='#EAEAEA').pack(anchor='w')
    elif type=="head":
        Label(chat_frame, image=img1, bg='#EAEAEA').pack(anchor='w')
    elif type=="tail":
        Label(chat_frame, image=img2, bg='#EAEAEA').pack(anchor='w')
    else:
        img3 =
ImageTk.PhotoImage(Image.open('assets/images/dice/'+type+'.jpg').resize((200,200),
Image.ANTIALIAS))
        Label(chat_frame, image=img3, bg='#EAEAEA').pack(anchor='w')

def showImages(query):
    global img0, img1, img2, img3

```

```

Label(imageContainer, image=img1, bg='#EAEAEA').grid(row=0, column=1)
Label(imageContainer, image=img2, bg='#EAEAEA').grid(row=1, column=0)
Label(imageContainer, image=img3, bg='#EAEAEA').grid(row=1, column=1)

##### WAEM - WhatsApp Email
#####
def sendWAEM():
    global rec_phoneno, rec_email
    data = WAEMEntry.get()
    rec_email, rec_phoneno = data, data
    WAEMEntry.delete(0, END)
    app_control.Win_Opt('close')
def send(e):
    sendWAEM()

def WAEMPOPUP(Service='None', rec='Reciever'):
    global WAEMEntry
    PopUProot = Tk()
    PopUProot.title(f'{Service} Service')
    PopUProot.configure(bg='white')

    if Service=="WhatsApp": PopUProot.iconbitmap("assets/images/whatsapp.ico")
    else: PopUProot.iconbitmap("assets/images/email.ico")
    w_width, w_height = 410, 200
    s_width, s_height = PopUProot.winfo_screenwidth(), PopUProot.winfo_screenheight()
    x, y = (s_width/2)-(w_width/2), (s_height/2)-(w_height/2)
    PopUProot.geometry('%dx%d+%d+%d' % (w_width,w_height,x,y-30)) #center location of
the screen
    Label(PopUProot, text=f'Reciever {rec}', font=('Arial', 16), bg='white').pack(pady=(20, 10))
    WAEMEntry = Entry(PopUProot, bd=10, relief=FLAT, font=('Arial', 12), justify='center',
bg='#DCDCDC', width=30)
    WAEMEntry.pack()
    WAEMEntry.focus()

    SendBtn = Button(PopUProot, text='Send', font=('Arial', 12), relief=FLAT, bg='#14A769',
fg='white', command=sendWAEM)
    SendBtn.pack(pady=20, ipadx=10)
    PopUProot.bind('<Return>', send)
    PopUProot.mainloop()

##### CHANGING CHAT BACKGROUND COLOR
#####
def getChatColor():
    global chatBgColor
    myColor = colorchooser.askcolor()
    if myColor[1] is None: return
    chatBgColor = myColor[1]
    colorbar['bg'] = chatBgColor
    chat_frame['bg'] = chatBgColor
    root1['bg'] = chatBgColor
    ChangeSettings(True)

```

```

TextModeFrame.pack(fill=BOTH)
    UserField.focus()
    chatMode=0
else:
    # appControl.volumeControl('full')
    TextModeFrame.pack_forget()
    VoiceModeFrame.pack(fill=BOTH)
    root.focus()
    chatMode=1

##### GUI
#####

def onhover(e):
    userPhoto['image'] = chngPh
def onleave(e):
    userPhoto['image'] = userProfileImg

def UpdateIMAGE():
    global ownerPhoto, userProfileImg, userIcon

    os.system('python modules/avatar_selection.py')
    u = UserData()
    u.extractData()
    ownerPhoto = u.getUserPhoto()
    userProfileImg =
ImageTk.PhotoImage(Image.open("assets/images/avatars/a"+str(ownerPhoto)+".png").resize((12
0, 120)))

    userPhoto['image'] = userProfileImg
    userIcon =
PhotoImage(file="assets/images/avatars/ChatIcons/a"+str(ownerPhoto)+".png")

def SelectAvatar():
    Thread(target=UpdateIMAGE).start()

##### MAIN GUI
#####

#### SPLASH/LOADING SCREEN ####
def progressbar():
    s = ttk.Style()
    s.theme_use('clam')
    s.configure("white.Horizontal.TProgressbar", foreground='white', background='white')
    progress_bar = ttk.Progressbar(splash_root,style="white.Horizontal.TProgressbar",
orient="horizontal",mode="determinate", length=303)
    progress_bar.pack()
    splash_root.update()
    progress_bar['value'] = 0
    splash_root.update()

```

```

if __name__ == '__main__':
    splash_root = Tk()
    splash_root.configure(bg='#3895d3')
    splash_root.overridedirect(True)
    splash_label = Label(splash_root, text="Processing...",
font=('montserrat',15),bg='#3895d3',fg='white')
    splash_label.pack(pady=40)
    # splash_percentage_label = Label(splash_root, text="0 %",
font=('montserrat',15),bg='#3895d3',fg='white')
    # splash_percentage_label.pack(pady=(0,10))

    w_width, w_height = 400, 200
    s_width, s_height = splash_root.winfo_screenwidth(), splash_root.winfo_screenheight()
    x, y = (s_width/2)-(w_width/2), (s_height/2)-(w_height/2)
    splash_root.geometry('%dx%d+%d+%d' % (w_width,w_height,x,y-30))

    progressbar()
    splash_root.after(10, destroySplash)
    splash_root.mainloop()

    root = Tk()
    root.title('F.R.I.D.A.Y')
    w_width, w_height = 400, 650
    s_width, s_height = root.winfo_screenwidth(), root.winfo_screenheight()
    x, y = (s_width/2)-(w_width/2), (s_height/2)-(w_height/2)
    root.geometry('%dx%d+%d+%d' % (w_width,w_height,x,y-30)) #center location of the
screen
    root.configure(bg=background)
    # root.resizable(width=False, height=False)
    root.pack_propagate(0)

    root1 = Frame(root, bg=chatBgColor)
    root2 = Frame(root, bg=background)
    root3 = Frame(root, bg=background)

    for f in (root1, root2, root3):
        f.grid(row=0, column=0, sticky='news')

    #####
    ##### CHAT SCREEN #####
    #####

    #Chat Frame
    chat_frame = Frame(root1, width=380,height=551,bg=chatBgColor)
    chat_frame.pack(padx=10)
    chat_frame.pack_propagate(0)

    bottomFrame1 = Frame(root1, bg='#dfdfdf', height=100)
    bottomFrame1.pack(fill=X, side=BOTTOM)
    VoiceModeFrame = Frame(bottomFrame1, bg='#dfdfdf')
    VoiceModeFrame.pack(fill=BOTH)
    TextModeFrame = Frame(bottomFrame1, bg='#dfdfdf')

```



```

cblDarkImg = PhotoImage(file='assets/images/centralButton1.png')
if KCS_IMG==1: cblimage=cblDarkImg
else: cblimage=cblLightImg
cbl = Label(VoiceModeFrame, fg='white', image=cblimage, bg='#dfdfdf')
cbl.pack(pady=17)
AITaskStatusLbl = Label(VoiceModeFrame, text=' Offline', fg='white',
bg=AITaskStatusLblBG, font=('montserrat', 16))
AITaskStatusLbl.place(x=140,y=32)

#Settings Button
sphLight = PhotoImage(file = "assets/images/setting.png")
sphLight = sphLight.subsample(2,2)
sphDark = PhotoImage(file = "assets/images/setting1.png")
sphDark = sphDark.subsample(2,2)
if KCS_IMG==1: sphimage=sphDark
else: sphimage=sphLight
settingBtn = Button(VoiceModeFrame,image=sphimage,height=30,width=30,
bg='#dfdfdf',borderwidth=0,activebackground="#dfdfdf",command=lambda: raise_frame(root2))
settingBtn.place(relx=1.0, y=30,x=-20, anchor="ne")

#Keyboard Button
kbphLight = PhotoImage(file = "assets/images/keyboard.png")
kbphLight = kbphLight.subsample(2,2)
kbphDark = PhotoImage(file = "assets/images/keyboard1.png")
kbphDark = kbphDark.subsample(2,2)
if KCS_IMG==1: kbphimage=kbphDark
else: kbphimage=kbphLight
kbBtn = Button(VoiceModeFrame,image=kbphimage,height=30,width=30,
bg='#dfdfdf',borderwidth=0,activebackground="#dfdfdf", command=changeChatMode)
kbBtn.place(x=25, y=30)

#Mic
micImg = PhotoImage(file = "assets/images/mic.png")
micImg = micImg.subsample(2,2)
micBtn = Button(TextModeFrame,image=micImg,height=30,width=30,
bg='#dfdfdf',borderwidth=0,activebackground="#dfdfdf", command=changeChatMode)
micBtn.place(relx=1.0, y=30,x=-20, anchor="ne")

#Text Field
TextFieldImg = PhotoImage(file='assets/images/textField.png')
UserFieldLBL = Label(TextModeFrame, fg='white', image=TextFieldImg, bg='#dfdfdf')
UserFieldLBL.pack(pady=17, side=LEFT, padx=10)
UserField = Entry(TextModeFrame, fg='white', bg='#203647', font=('Montserrat', 16),
bd=6, width=22, relief=FLAT)
UserField.place(x=20, y=30)
UserField.insert(0, "Ask me anything...")
UserField.bind('<Return>', keyboardInput)

#User and Bot Icon
userIcon =
PhotoImage(file="assets/images/avatars/ChatIcons/a"+str(ownerPhoto)+".png")

```

```

settingsLbl = Label(root2, text='Settings', font=('Arial Bold', 15), bg=background, fg=textColor)
settingsLbl.pack(pady=10)
separator = ttk.Separator(root2, orient='horizontal')
separator.pack(fill=X)
#User Photo
userProfileImg = Image.open("assets/images/avatars/a"+str(ownerPhoto)+".png")
userProfileImg = ImageTk.PhotoImage(userProfileImg.resize((120, 120)))
userPhoto = Button(root2, image=userProfileImg, bg=background, bd=0, relief=FLAT,
activebackground=background, command=SelectAvatar)
userPhoto.pack(pady=(20, 5))

#Change Photo
chgPh =
ImageTk.PhotoImage(Image.open("assets/images/avatars/changephoto2.png").resize((120,
120)))

userPhoto.bind('<Enter>', onhover)
userPhoto.bind('<Leave>', onleave)

#Username
userName = Label(root2, text=ownerName, font=('Arial Bold', 15), fg=textColor,
bg=background)
userName.pack()

#Settings Frame
settingsFrame = Frame(root2, width=300, height=300, bg=background)
settingsFrame.pack(pady=20)

assLbl = Label(settingsFrame, text='Assistant Voice', font=('Arial', 13), fg=textColor,
bg=background)
assLbl.place(x=0, y=20)
n = StringVar()
assVoiceOption = ttk.Combobox(settingsFrame, values=('Female', 'Male'), font=('Arial',
13), width=13, textvariable=n)
assVoiceOption.current(voice_id)
assVoiceOption.place(x=150, y=20)
assVoiceOption.bind('<<ComboboxSelected>>', changeVoice)

voiceRateLbl = Label(settingsFrame, text='Voice Rate', font=('Arial', 13), fg=textColor,
bg=background)
voiceRateLbl.place(x=0, y=60)
n2 = StringVar()
voiceOption = ttk.Combobox(settingsFrame, font=('Arial', 13), width=13,
textvariable=n2)
voiceOption['values'] = ('Very Low', 'Low', 'Normal', 'Fast', 'Very Fast')
voiceOption.current(ass_voiceRate//50-2) #100 150 200 250 300
voiceOption.place(x=150, y=60)
voiceOption.bind('<<ComboboxSelected>>', changeVoiceRate)

volumeLbl = Label(settingsFrame, text='Volume', font=('Arial', 13), fg=textColor,
bg=background)

```

```

darkBtn = ttk.Radiobutton(settingsFrame, text='Dark', value=1, variable=themeValue,
style='Wild.TRadiobutton', command=changeTheme, takefocus=False)
    darkBtn.place(x=150,y=145)
    lightBtn = ttk.Radiobutton(settingsFrame, text='Light', value=2, variable=themeValue,
style='Wild.TRadiobutton', command=changeTheme, takefocus=False)
    lightBtn.place(x=230,y=145)
    themeValue.set(1)
    if KCS_IMG==0: themeValue.set(2)


    chooseChatLbl = Label(settingsFrame, text='Chat Background', font=('Arial', 13),
fg=textColor, bg=background)
    chooseChatLbl.place(x=0,y=180)
    cimg = PhotoImage(file = "assets/images/colorchooser.png")
    cimg = cimg.subsample(3,3)
    colorbar = Label(settingsFrame, bd=3, width=18, height=1, bg=chatBgColor)
    colorbar.place(x=150, y=180)
    if KCS_IMG==0: colorbar['bg'] = '#E8EBEF'
    Button(settingsFrame, image=cimg, relief=FLAT, command=getChatColor).place(x=261,
y=180)


    backBtn = Button(settingsFrame, text=' Back ', bd=0, font=('Arial 12'), fg='white',
bg='#14A769', relief=FLAT, command=lambda:raise_frame(root1))
    clearFaceBtn = Button(settingsFrame, text=' Clear Facial Data ', bd=0, font=('Arial 12'),
fg='white', bg='#14A769', relief=FLAT, command=deleteUserData)
    backBtn.place(x=5, y=250)
    clearFaceBtn.place(x=120, y=250)


    try:
        # pass
        Thread(target=voiceMedium).start()
    except:
        pass
    try:
        # pass
        Thread(target=web_scrapping.dataUpdate).start()
    except Exception as e:
        print('System is Offline...')


    root.iconbitmap('assets/images/assistant2.ico')
    raise_frame(root1)
    root.mainloop()

```

### math\_function.py

```
import math
```

```

def basicOperations(text):
    if 'root' in text:
        temp = text.rfind(' ')
        num = int(text[temp+1:])
        return round(math.sqrt(num),2)

```

```

text = text.replace('plus', '+')
text = text.replace('minus', '-')
text = text.replace('x', '*')
text = text.replace('multiplied by', '*')
text = text.replace('multiply', '*')
text = text.replace('divided by', '/')
text = text.replace('to the power', '**')
text = text.replace('power', '**')
result = eval(text)
return round(result,2)

```

```

def bitwiseOperations(text):
    if 'right shift' in text:
        temp = text.rfind(' ')
        num = int(text[temp+1:])
        return num>>1
    elif 'left shift' in text:
        temp = text.rfind(' ')
        num = int(text[temp+1:])
        return num<<1
    text = text.replace('and', '&')
    text = text.replace('or', '|')
    text = text.replace('not of', '~')
    text = text.replace('not', '~')
    text = text.replace('xor', '^')
    result = eval(text)
    return result

```

```

def conversions(text):
    temp = text.rfind(' ')
    num = int(text[temp+1:])
    if 'bin' in text:
        return eval('bin(num)')[2:]
    elif 'hex' in text:
        return eval('hex(num)')[2:]
    elif 'oct' in text:
        return eval('oct(num)')[2:]

```

```

def trigonometry(text):
    temp = text.replace('degree', '')
    temp = text.rfind(' ')
    deg = int(text[temp+1:])
    rad = (deg * math.pi) / 180
    if 'sin' in text:
        return round(math.sin(rad),2)
    elif 'cos' in text:
        return round(math.cos(rad),2)
    elif 'tan' in text:
        return round(math.tan(rad),2)

```

```

def factorial(n):

```

```

else: return n*factorial(n-1)

def logFind(text):
    temp = text.rfind(' ')
    num = int(text[temp+1:])
    return round(math.log(num,10),2)

def isHaving(text, lst):
    for word in lst:
        if word in text:
            return True
    return False

def perform(text):
    text = text.replace('math','')
    if "factorial" in text: return str(factorial(int(text[text.rfind('')+1:])))
    elif isHaving(text, ['sin','cos','tan']): return str(trigonometry(text))
    elif isHaving(text, ['bin','hex','oct']): return str(conversions(text))
    elif isHaving(text, ['shift','and','or','not']): return str(bitwiseOperations(text))
    elif 'log' in text: return str(logFind(text))
    else: return str(basicOperations(text))

```

### normal\_chat.py

```

from difflib import get_close_matches
import json
from random import choice
import datetime

class DateTime:
    def currentTime(self):
        time = datetime.datetime.now()
        x = " A.M."
        if time.hour>12: x = " P.M."
        time = str(time)
        time = time[11:16] + x
        return time

    def currentDate(self):
        now = datetime.datetime.now()
        day = now.strftime('%A')
        date = str(now)[8:10]
        month = now.strftime('%B')
        year = str(now.year)
        result = f'{day}, {date} {month}, {year}'
        return result

    def wishMe():
        now = datetime.datetime.now()
        hr = now.hour
        if hr<12:

```

```

from PIL import Image, ImageTk
import cv2
import numpy as np
import os
from os.path import isfile, join
from threading import Thread
from .user_handler import UserData
from . import face_unlocker as FU

background, textColor = 'black', '#F6FAFB'
background, textColor = textColor, background

avatarChosen = 0
chosenAvtrImage = None
user_name = ""
user_gender = ""

try:
    face_classifier = cv2.CascadeClassifier('Cascade/haarcascade_frontalface_default.xml')
except Exception as e:
    print('Cascade File is missing...')
    raise SystemExit

if os.path.exists('userData')==False:
    os.mkdir('userData')
if os.path.exists('userData/faceData')==False:
    os.mkdir('userData/faceData')

##### ROOT1 #####
def startLogin():
    try:
        result = FU.startDetecting()
        if result:
            user = UserData()
            user.extractData()
            userName = user.getName().split()[0]
            welcLbl['text'] = 'Hi '+userName+',\nWelcome to the world of\nScience
& Technology'

            loginStatus['text'] = 'UNLOCKED'
            loginStatus['fg'] = 'green'
            faceStatus['text']=(Logged In)
            os.system('python modules/gui_assistant.py')
        else:
            print('Error Occurred')

    except Exception as e:
        print(e)

##### ROOT2 #####

```

```

def trainFace():
    data_path = 'userData/faceData/'
    onlyfiles = [f for f in os.listdir(data_path) if isfile(join(data_path, f))]

    Training_data = []
    Labels = []

    for i, files in enumerate(onlyfiles):
        image_path = data_path + onlyfiles[i]
        images = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

        Training_data.append(np.asarray(images, dtype=np.uint8))
        Labels.append(i)

    Labels = np.asarray(Labels, dtype=np.int32)

    model = cv2.face.LBPHFaceRecognizer_create()
    model.train(np.asarray(Training_data), np.asarray(Labels))

    print('Model Trained Successfully !!!')
    model.save('userData/trainer.yml')
    print('Model Saved !!!')

def face_extractor(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray, 1.3, 5)

    if faces is ():
        return None

    for (x, y, w, h) in faces:
        cropped_face = img[y:y+h, x:x+w]

    return cropped_face

cap = None
count = 0
def startCapturing():
    global count, cap
    ret, frame = cap.read()
    if face_extractor(frame) is not None:
        count += 1
        face = cv2.resize(face_extractor(frame), (200, 200))
        face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)

        file_name_path = 'userData/faceData/img' + str(count) + '.png'
        cv2.imwrite(file_name_path, face)
        print(count)
        progress_bar['value'] = count

```

```

if count==100:
    progress_bar.destroy()
    lmain['image'] = defaultImg2
    statusLbl['text'] = '(Face added successfully)'
    cap.release()
    cv2.destroyAllWindows()
    Thread(target=trainFace).start()
    addBtn['text'] = '    Next    '
    addBtn['command'] = lambda:raise_frame(root3)
    return

    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
    frame = cv2.flip(frame, 1)
    img = Image.fromarray(frame)
    imgtk = ImageTk.PhotoImage(image=img)
    lmain.imgtk = imgtk
    lmain.configure(image=imgtk)
    lmain.after(10, startCapturing)

def Add_Face():

    global cap, user_name, user_gender
    user_name = nameField.get()
    user_gender = r.get()
    if user_name != "" and user_gender!=0:
        if agr.get()==1:
            cap = cv2.VideoCapture(0)
            startCapturing()
            progress_bar.place(x=20, y=273)
            statusLbl['text'] = ""

        else:
            statusLbl['text'] = '(Check the Condition)'

    else:
        statusLbl['text'] = '(Please fill the details)'

def SuccessfullyRegistered():
    if avatarChosen != 0:
        gen = 'Male'
        if user_gender==2: gen = 'Female'
        u = UserData()
        u.updateData(user_name, gen, avatarChosen)
        usernameLbl['text'] = user_name
        raise_frame(root4)

def selectAVATAR(avt=0):
    global avatarChosen, choosedAvtrImage
    avatarChosen = avt
    i=1
    for avtr in (avtb1,avtb2,avtb3,avtb4,avtb5,avtb6,avtb7,avtb8):
        if i==avt:
            avtr['state'] = 'disabled'

```



```

#####
##### FACE ADD SCREEN #####
#####

image2 = Image.open('assets/images/defaultFace4.png')
image2 = image2.resize((300, 250))
defaultImg2 = ImageTk.PhotoImage(image2)

dataFrame2 = Frame(root2, bd=10, bg=background)
dataFrame2.pack(fill=X)
lmain = Label(dataFrame2, width=300, height=250, image=defaultImg2)
lmain.pack(padx=10, pady=10)

#Details
detailFrame2 = Frame(root2, bd=10, bg=background)
detailFrame2.pack(fill=X)
userFrame2 = Frame(detailFrame2, bd=10, width=300, height=250, relief=FLAT, bg=background)
userFrame2.pack(padx=10, pady=10)

#progress
progress_bar = ttk.Progressbar(root2, orient=HORIZONTAL, length=303, mode='determinate')

#name
nameLbl = Label(userFrame2, text='Name', font=('Arial Bold', 12), fg='#303E54', bg=background)
nameLbl.place(x=10,y=10)
nameField = Entry(userFrame2, bd=5, font=('Arial Bold', 10), width=25, relief=FLAT,
bg='#D4D5D7')
nameField.focus()
nameField.place(x=80,y=10)

genLbl = Label(userFrame2, text='Gender', font=('Arial Bold', 12), fg='#303E54', bg=background)
genLbl.place(x=10,y=50)
r = IntVar()
s = ttk.Style()
s.configure('Wild.TRadiobutton', background=background, foreground=textColor, font=('Arial
Bold', 10), focuscolor=s.configure(".").["background"])
genMale = ttk.Radiobutton(userFrame2, text='Male', value=1, variable=r,
style='Wild.TRadiobutton', takefocus=False)
genMale.place(x=80,y=52)
genFemale = ttk.Radiobutton(userFrame2, text='Female', value=2, variable=r,
style='Wild.TRadiobutton', takefocus=False)
genFemale.place(x=150,y=52)

#agreement
agr = IntVar()
sc = ttk.Style()
sc.configure('Wild.TCheckbutton', background=background, foreground='#303E54', font=('Arial
Bold',10), focuscolor=sc.configure(".").["background"])
# agree = Checkbutton(userFrame2, text='I agree to use my face for Security purpose',
fg=textColor, bg=background, activebackground=background, activeforeground=textColor)
agree = ttk.Checkbutton(userFrame2, text='I agree to use my Face for Security',

```

```
#####  
#### AVATAR SELECTION ####  
#####
```

```
Label(root3, text="Choose Your Avatar", font=('arial', 15), bg=background, fg='#303E54').pack()
```

```
avatarContainer = Frame(root3, bg=background, width=300, height=500)  
avatarContainer.pack(pady=10)  
size = 100
```

```
avtr1 = Image.open('assets/images/avatars/a1.png')  
avtr1 = avtr1.resize((size, size))  
avtr1 = ImageTk.PhotoImage(avtr1)  
avtr2 = Image.open('assets/images/avatars/a2.png')  
avtr2 = avtr2.resize((size, size))  
avtr2 = ImageTk.PhotoImage(avtr2)  
avtr3 = Image.open('assets/images/avatars/a3.png')  
avtr3 = avtr3.resize((size, size))  
avtr3 = ImageTk.PhotoImage(avtr3)  
avtr4 = Image.open('assets/images/avatars/a4.png')  
avtr4 = avtr4.resize((size, size))  
avtr4 = ImageTk.PhotoImage(avtr4)  
avtr5 = Image.open('assets/images/avatars/a5.png')  
avtr5 = avtr5.resize((size, size))  
avtr5 = ImageTk.PhotoImage(avtr5)  
avtr6 = Image.open('assets/images/avatars/a6.png')  
avtr6 = avtr6.resize((size, size))  
avtr6 = ImageTk.PhotoImage(avtr6)  
avtr7 = Image.open('assets/images/avatars/a7.png')  
avtr7 = avtr7.resize((size, size))  
avtr7 = ImageTk.PhotoImage(avtr7)  
avtr8 = Image.open('assets/images/avatars/a8.png')  
avtr8 = avtr8.resize((size, size))  
avtr8 = ImageTk.PhotoImage(avtr8)
```

```
avtb1 = Button(avatarContainer, image=avtr1, bg=background, activebackground=background,  
relief=FLAT, bd=0, command=lambda:selectAVATAR(1))  
avtb1.grid(row=0, column=0, ipadx=25, ipady=10)
```

```
avtb2 = Button(avatarContainer, image=avtr2, bg=background, activebackground=background,  
relief=FLAT, bd=0, command=lambda:selectAVATAR(2))  
avtb2.grid(row=0, column=1, ipadx=25, ipady=10)
```

```
avtb3 = Button(avatarContainer, image=avtr3, bg=background, activebackground=background,  
relief=FLAT, bd=0, command=lambda:selectAVATAR(3))  
avtb3.grid(row=1, column=0, ipadx=25, ipady=10)
```

```
avtb4 = Button(avatarContainer, image=avtr4, bg=background, activebackground=background,  
relief=FLAT, bd=0, command=lambda:selectAVATAR(4))  
avtb4.grid(row=1, column=1, ipadx=25, ipady=10)
```

```

avtb7 = Button(avatarContainer, image=avtr7, bg=background, activebackground=background,
relief=FLAT, bd=0, command=lambda:selectAVATAR(7))
avtb7.grid(row=3, column=0, ipadx=25, ipady=10)

avtb8 = Button(avatarContainer, image=avtr8, bg=background, activebackground=background,
relief=FLAT, bd=0, command=lambda:selectAVATAR(8))
avtb8.grid(row=3, column=1, ipadx=25, ipady=10)

Button(root3, text='    Submit    ', font=('Arial Bold', 15), bg='#01933B', fg='white', bd=0,
relief=FLAT, command=SuccessfullyRegistered).pack()

#####
##### SUCCESSFULL REGISTRATION #####
#####

userPIC = Label(root4, bg=background, image=avtr1)
userPIC.pack(pady=(40, 10))
usernameLbl = Label(root4, text="Roshan Kumar", font=('Arial Bold',15), bg=background,
fg='#85AD4F')
usernameLbl.pack(pady=(0, 70))

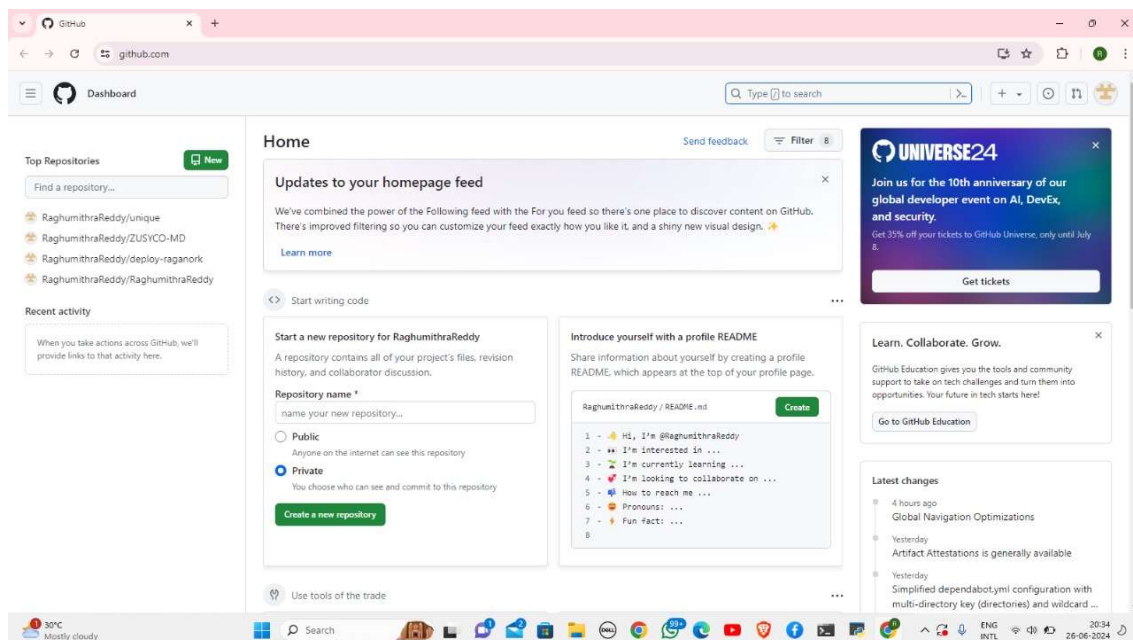
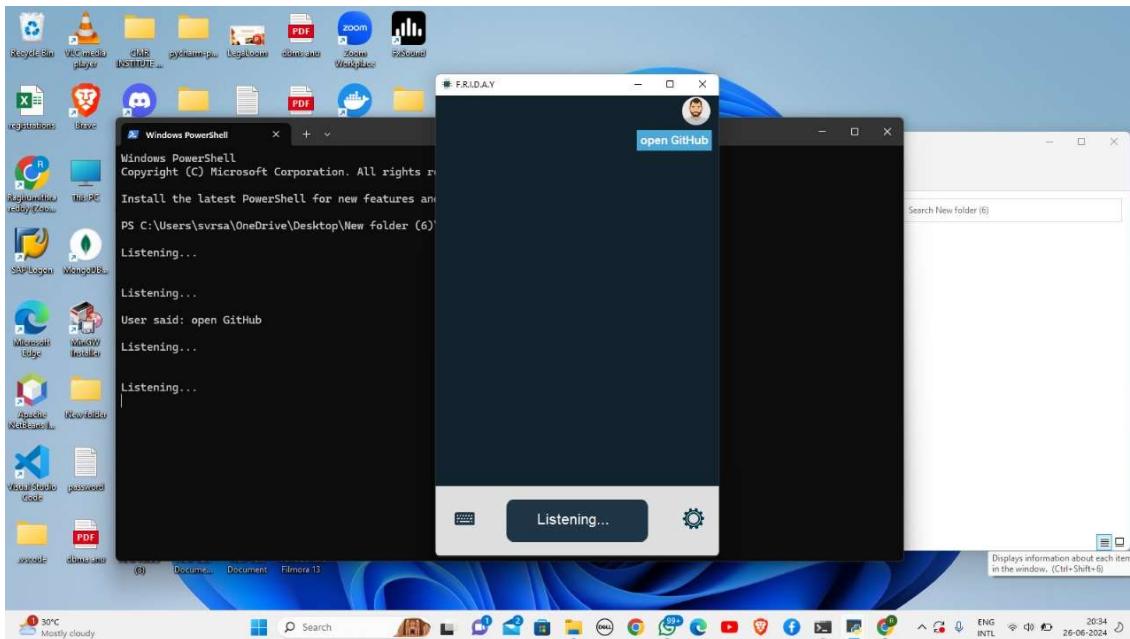
Label(root4, text="Your account has been successfully activated!", font=('Arial Bold',15),
bg=background, fg='#303E54', wraplength=300).pack(pady=10)
Label(root4, text="Launch the APP again to get started the conversation with your Personal
Assistant", font=('arial',13), bg=background, fg='#A3A5AB', wraplength=350).pack()

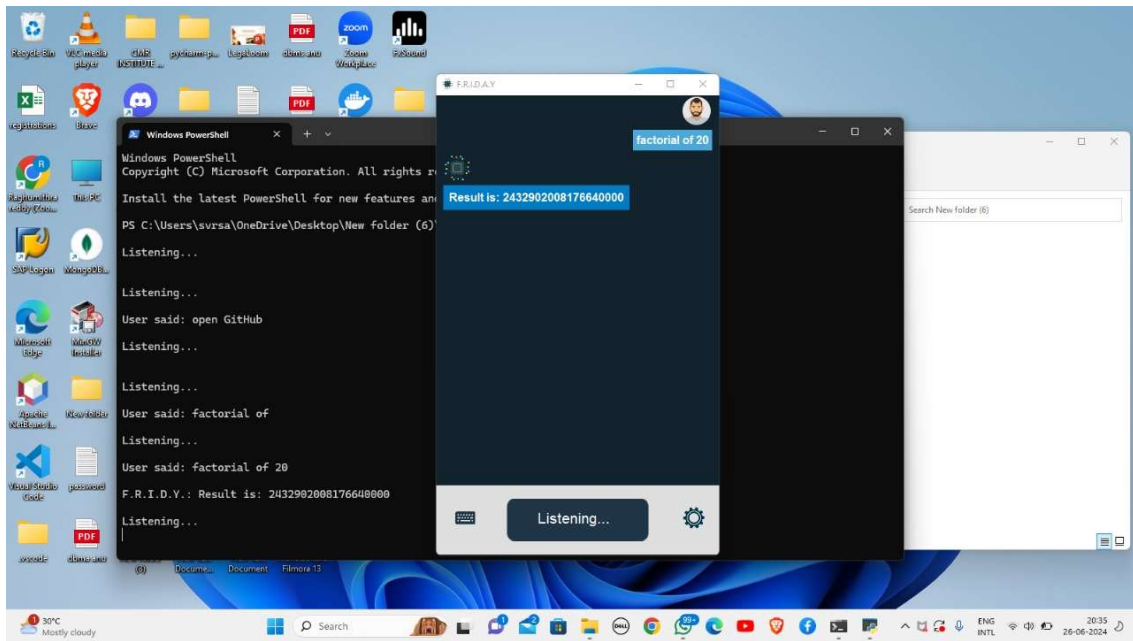
Button(root4, text='    OK    ', bg='#0475BB', fg='white',font=('Arial Bold', 18), bd=0, relief=FLAT,
command=lambda:quit()).pack(pady=50)

root.iconbitmap('assets/images/assistant2.ico')
raise_frame(root1)
root.mainloop()

```

# OUTPUT SCREENSHOTS





## CONCLUSION

In conclusion, the Personal Chatbot with Face Lock Detection project is a pioneering effort that showcases the potential of integrating cutting-edge technologies to create a secure, personalized, and interactive user experience. By leveraging the power of computer vision and natural language processing, this project demonstrates a novel approach to chatbot development that prioritizes user security and convenience.

The face lock detection module, built using OpenCV, provides a robust and accurate means of verifying user identities, ensuring that only authorized individuals can access the chatbot. This feature is particularly significant in today's digital landscape, where data privacy and security are paramount concerns.

Meanwhile, the chatbot module, powered by NLTK, offers a highly responsive and personalized experience, allowing users to interact with the system in a natural and intuitive way. The chatbot's ability to understand and respond to user queries in a contextually relevant manner sets a new standard for chatbot interactions.

The successful integration of these two modules is a testament to the project's innovative approach and technical expertise. By combining the strengths of computer vision and natural language processing, this project has created a truly unique and compelling user experience that has far-reaching implications for various industries, including healthcare, finance, and customer service.

Furthermore, this project highlights the potential for AI-powered systems to improve user engagement, increase efficiency, and enhance overall user satisfaction. As the technology continues to evolve, we can expect to see even more sophisticated and personalized interactions between humans and machines.

In summary, the Personal Chatbot with Face Lock Detection project is a groundbreaking achievement that demonstrates the power of innovation and collaboration in creating cutting-edge technologies that can transform the way we live and work.

## References

<https://medium.com/nerd-for-tech/build-your-personal-chatgpt-bot-with-streamlit-and-openai-apis-84a05ab6929b>

<https://arxiv.org/pdf/2201.06657v1>

<https://medium.com/@JeffyJeff/a-step-by-step-guide-to-creating-your-own-assistant-chatbot-using-openais-assistant-api-and-react-655391215c3a>

[https://www.researchgate.net/publication/354279472\\_An\\_Artificial\\_Intelligence\\_Based\\_Virtual\\_Assistant\\_Using\\_Conversational\\_Agents](https://www.researchgate.net/publication/354279472_An_Artificial_Intelligence_Based_Virtual_Assistant_Using_Conversational_Agents)